DOE/MC/29113--4048
EM report.

Integrated Computer-Enhanced Remote Viewing System

Quarterly Report Number 5
for
October - December 1993

February 22, 1994

Work Performed under
Contract No.:DE-AC21-92MC29113

For:
U.S. Department of Energy
Morgantown Energy Technology Center
3610 Collins Ferry Road
Morgantown, West Virginia 26507

By:
Mechanical Technology Incorporated
968 Albany-Shaker Road
Latham, New York 12110
Tel No. (518) 785 - 2800

1. Introduction

The Interactive, Computer-Enhanced, Remote Viewing System (ICERVS) is a system designed to provide a reliable geometric description of a robotic task space in a fashion that enables robotic remediation to be carried out more efficiently and economically that with present systems. The key elements are a faithful way to store empirical data and a friendly user interface that provides an operator with timely access to all that is known about a scene.

ICERVS will help an operator to analyze a scene and generate additional geometric data for automating significant portions of the remediation activity. Features that enable this include the following:

- Storage and display of empirical sensor data,
- Ability to update segments of the geometric description of the task space,
- Side-by-side comparisons of a live TV scene and a computer generated view of the same scene,
- Ability to create and display computer models of perceived objects in the task space, together with textual comments, and
- Easy export of data to robotic world models for robot guidance.

The development of ICERVS is to occur in three phases.

- Phase 1 will focus on the development of the Data Library, which contains the geometric data about the task space and the objects in it, and the Toolkit, which includes the mechanisms for manipulating and displaying both empirical and model data.
- Phase 2 will concentrate on integrating these subsystems with a sensor subsystem into one working system. Some additional functionality will be incorporated in the Data Library and Toolkit subsystems.
- Phase 3 will expand the configuration to meet the needs of a full scale demonstration of the interactive mapping of some waste site to be identified.

The Phase 1 activities ended in June and the Phase 2 contract modification was authorized on September 24, 1993. Work on Phase 2 was unofficially begun in August, when the intent to proceed to Phase 2 was clear. This was

done in order to meet the schedule of the Buried Waste Integrated Demonstration to be carried out at the Idaho National Engineering Laboratory (INEL) in the fourth quarter of FY94.

This report covers the technical effort carried out during the quarter consisting of October, November and December 1993.

2. Summary of Progress to Date

The second Phase of the ICERVS project consists of nine tasks which are described in Attachment A. This listing includes the eight tasks originally proposed together with an additional task to supply a working copy of ICERVS to INEL for use with the Buried Waste Integrated Demonstration (BWID).

Significant efforts were devoted to Task 2 and the detailed design of the Common Interface for Sensors. This effort is running in parallel with an internal effort to complete a CRADA development of the structured light sensor system which will be included in the Phase 2 demonstration of ICERVS.

Significant efforts were devoted to the completion of Task 4: Analysis Software Design. The Phase 2 software architecture from Task 1 (Intermediate System Design) has been expanded and detailed for the Analysis Software. Software classes have been defined and detailed for the Demonstration Application CSCI and the Volumetric Data Subsystem CSCI. A draft of the Subsystem Design Report has been prepared and included with this report as Attachment B.

Some initial efforts were directed at starting the Analysis Software implementation. Several utility routines from Phase I were upgraded to reflect changes in the Rogue Wave C++ libraries. The main window and main menu codes for the Demonstration Application CSCI and Volumetric Data Subsystem have been implemented and integrated to produce a preliminary test platform for the development of the user interface software components.

3. Progress Report

3.1 Task 2 – Sensor Subsystem

The sensor subsystem is the major addition to the scope of the Phase 1 system. The sensor subsystem represents the source of all input data and, for Phase 2, contains a structured light mapping sensor, a laser range finder mapping sensor, and a collection of sensors that measure physical properties

of the material (e.g., temperature, conductivity, radiation level) whose data will be provided to ICERVS through the Minilab subsystem being developed by Sandia National Laboratory (SNL). Architectural data about a plant or factory will also enter ICERVS through the software mechanism associated with the sensor subsystem.

The structured light sensor system is currently under development by MTI as part of a CRADA with both SNL and Oak Ridge National Laboratory (ORNL). A demonstration version of the sensor will be demonstrated in mid-February 1994. This system will be demonstrated operating in a stand-alone mode, but will be capable of remote operation as an ICERVS subsystem. MTI will make this sensor system available to the project to carry out the Phase 2 development.

During the current quarter details for the Common Interface For Sensors have been generated. The command set for the Structured Light Sensor Server has been identified and forwarded to ORNL. Interface requirements between ICERVS and the CRADA sensor have been identified and incorporated into the design of the sensor software. Data output formats of the ORNL user interface code are being reviewed as a first step toward defining the input formats for ICERVS.

3.2 Task 4 – Analysis Software Design.

The analysis software design effort began with the System Design Report produced in Task 1 and proceeded add to additional detail. The design effort was divided into several sections: Client-Server Communications Design, Demonstration Application CSCI Design, and Volumetric Data CSCI Design. Each of these design efforts is described in later paragraphs.

The effort on Task 4 concluded with the preparation of the sections 1.0 through 7.0 of the Subsystem Design Report which documents the Analysis Software Design. A draft release of that report is included with this document as Attachment B.

Due the large volume of design information needed to describe the Demonstration Application and Volumetric Data CSCIs, it was decided to provide sections 8.0 through 11.0 (the Sensor Interface and Sensor CSCIs) in a separate report which is expected to be included in the next Technical Quarterly.

3.2.1 Client-Server Communications Design

The general design of ICERVS is that of one or more client processes (Demonstration Application, Volumetric Data Subsystem User Interface, etc.) that require connection to one or more server processes (VDS, sensors, etc.). The clients may reside on the same computer system as the servers or

different systems. UNIX sockets using Internet Protocol (IP) will be used for communications. In addition, in order to attain GISC compatibility, the Sandia developed GENISAS and ISOE software are being evaluated to serve as the underlying communications protocol.

The design effort began with the definition of command message and communications message protocols. A general implementation approach was developed which calls for initially using the CRADA developed communications software and gradually migrating to the ISOE and GENISAS packages by the end of Phase 3. Details of the design of the client-server communications are contained in section 5.0 of the ICERVS Phase 2 Subsystem Design Report.

3.2.2 Demonstration Application CSCI Design

The ICERVS Phase 2 Demonstration CSCI is the topmost application process and contains the high-level operator interface to demonstrate the functions and features of the ICERVS Phase II system. This CSCI is representative of a user application and may be used as a template for creating user applications. This CSCI is composed of a single Computer Software Component (CSC);

An object-oriented design methodology was used to design this CSCI. After defining the problem domain, the key problem domain component (PDC) classes were identified, including waste site, sensor, camera, dataset, property type, and view. The relationships, interactions, and associations among these classes were examined and an OOA diagram developed for the CSCI. Software classes corresponding to the PDC classes were then defined, OOD diagrams were developed, and finally, the major CSCI functions were described. A similar process was conducted to design the user interface elements of this CSCI.

The details of the design of the Demonstration Application CSCI are contained in section 6.0 of the ICERVS Phase 2 Subsystem Design Report.

3.2.3 Volumetric Data CSCI

The Volumetric Data CSCI encapsulates the storage, retrieval, manipulation, and visualization of spatial, property and geometric object data. It is composed of seven Computer Software Components (CSC).

An object-oriented design methodology was used to design this CSCI. After defining the problem domain, the key problem domain component (PDC) classes were identified, including waste site, property type, dataset, view, cutplane, geometric object, dimensional data, property data, and sensor data. The relationships, interactions, and associations among these classes were examined and an OOA diagram developed for the CSCI. Each CSC was then

addressed separately. An OOA diagram, software classes, OOD diagrams, and major function descriptions were developed for each CSC.

The design of the Volumetric Data CSCI is detailed in section 7.0 of the ICERVS Phase 2 Subsystem Design Report.

3.3 Task 5 – Analysis Software Code / Test.

Implementation of the analysis software has begun. The utility software classes from Phase I have been updated to reflect changes in the Rogue Wave libraries. An interface between the UIMX generated code and the ICERVS C++ code has be implemented. Software classes for the Demonstration Application CSCI and Volumetric Data CSCI top level window and menu systems have been implemented. Several utility software classes (browsers, editors, selection lists, etc.) have been implemented, interfaced to their low-level UIMX codes and integrated with the top level menu software.

Two executable programs has been implemented to serve as test platforms for the remaining software implementation. Program ICERVS2 implements the Demonstration Application CSCI top level window and menu. Many of the SYSTEM and SITE menu functions have been implemented, but the remaining menu functions perform no action at this time. Program VDSMAINU implements the Volumetric Data CSCI top level window and menu. None of the menu functions perform any actions at this time.

4. Plans for Future Activity

The work on Task 2: Sensor Subsystem is continuing and is being coordinated with the efforts to complete the CRADA mapping system for its demonstration scheduled for February 15,1994. The sensor system interface design will be completed in January and integration of the CRADA sensor with the ICERVS software will begin after the February demonstration.

Analysis Software Code / Test (Task 5) will continue throughout the next quarter. By the end of the quarter, most of the user interface should be operational, the client-server communications interface will be implemented, and the integration of TrueSolid will have begun.

5. Assessment of Prospects

The results after the second quarter of activity continue to be quite encouraging. Discussions with end users at various remediation sites reveal ongoing and increasing interest in ICERVS. This Phase will address the needs of the Buried Waste Integrated Demonstration (BWID), while a joint effort with another PRDA project in robotics will begin to explore the utility of ICERVS in the D&D application.

6. Appendices

A. Task Descriptions

B. Draft of Subsystem Design Report

Attachment A. ICERVS Task Description

ATTACHMENT A
ICERVS Task Description

The nine tasks in Phase II are described below.

Task 1: Intermediate System Design

Based on the results from Phase I, MTI will review and update the requirements for the ICERVS, including an evaluation of the "TrueSolid" software and alternative data formats to IGES. The result will be a modified set of requirements for integrating the various subsystems. MTI will complete and initial draft of the ICERVS generic sensor interface. MTI will also establish user focus groups in the DOE end-user community.

Task 2: Sensor Subsystem

MTI will make available a sensor subsystem developed under a CRADA with two national laboratories. This sensor subsystem has two instrument stations. Each station consists of a solid-state camera and a laser projector. Each station will also include the appropriate suspension and positioning hardware for investigating a simulated single shell tank and will provide for yaw and pitch motions.

MTI will upgrade the software in the structured light sensor to operate as an ICERVS sensor subsystem. This will involve defining a remote interface (in accordance with the ICERVS generic sensor interface) and developing the necessary hardware and software for integration with the analysis subsystem. MTI will also define appropriate interfaces for a laser range finder and Minilab. In the absence of empirical sensor data from such sensors, MTI will create simple simulators to test these interfaces.

Task 3: Computer Upgrade

MTI will upgrade the computing platform to enhance the functionality of the overall system. MTI will include, at least: a color TV video monitor, an upgraded central processing unit, and if necessary, additional internal memory. The Octree Corporation's "TrueSolid" software module (or equivalent) to facilitate the Octree engine shall also be included.

Task 4: Analysis Software Design

MTI will review and modify the analysis software requirements as identified in Task 1, Phase II. MTI will design software modules which implement a generic sensor interface, integrate the "TrueSolid" package, and which enhance the capabilities to include, but not be limited to: the ability to accept and store real empirical data (to encompass material properties); the ability to store and merge object models alongside the data they represent; the ability to convert data into a suitable format, such as IGES or STEP, to facilitate interaction with robotic controls. MTI will also design two sets of graphics tools. The first shall enhance the system's display capabilities, and the second shall improve the system's ability to build three-dimensional geometric models.

Task 5: Analysis Software Code and Unit Test

MTI will code, test, and integrate the new analysis software and test it in a stand-alone mode to verify functionality. The new analysis software will incorporate data library functions, interface to "TrueSolid", generic sensor interface, and data output in IGES or an alternative format.

Task 6: System Integration

MTI will integrate, install, and operate data library and toolkit analysis software in an interactive mode to ensure compatibility and interoperability.

MTI will design the software elements needed to ensure effective communication between the sensor subsystems and computing platform. MTI will write, test, and integrate the necessary computer codes required for data acquisition and interactive viewing.

MTI will design an operator interface which provides effective control of the structured light sensor subsystem (i.e. cameras, lights, and positioning systems). MTI will prepare a computer code which allows the operator to control the video camera view selection and simultaneously display a matching computer image.

MTI will interconnect all system components and verify their joint functionality. A simulated, single-shell tank shall also be prepared for demonstrating the subscale ICERVS.

Task 7: Demonstration

MTI will demonstrate and verify, at least, the following components: a sensor subsystem which automatically and interactively maps the surface and walls of a simulated single-shell tank; a sensor interface to a laser range finder; a sensor interface to a minilab subsystem; library analysis software which properly creates and manipulates an equivalent world model in a form compatible with the needs of a robotic controller; analysis software which also demonstrates enhanced viewing and model-building tools; and an ICERVS which demonstrates the ability to match scenes from live TV and the world model as the operator pans the camera.

Task 8: Buried Waste Application

Subtask 8.1: System Design Modifications

MTI will finalize the features and performance needed for the buried waste application through discussions with INEL personnel. MTI will perform design modifications of the ICERVS architecture and algorithms as needed.

Subtask 8.2: Detailed Design

MTI will design the analysis software elements to support the buried waste application, including: enhanced user interface, means for storage and display of

multiple material properties, updating of data from excavation activities, tools for registering data from different sensors, interpolation of material properties for analysis and display, and combination of multiple property data via boolean operations.

Subtask 8.3: Code and Unit Test

MTI will generate the necessary software to implement the features to support the buried waste application. Each element of code shall be unit tested and then tested again at the various levels of integration.

Subtask 8.4: Demonstration, Installation, and Training

MTI will incorporate the buried waste features as part of the ICERVS Phase II demonstration. MTI will install a copy of the ICERVS executable software on an INEL-supplied workstation, provide a user's guide, and supply a one-week level of effort for informal training at INEL.

Task 9: Topical Report and Decision Point

MTI will submit a topical report within sixty (60) days prior to completion of Phase II detailing the work completed to date and including the results of the Task 8 Buried Waste Application. Within thirty (30) days after submittal, the COR shall accept the draft topical report or recommend changes.

Within (3) days after submittal of the topical report, the contracting officer shall decide whether to proceed to the next phase. If the contracting officer decides NOT to continue to the next phase, MTI will submit a camera-ready final copy of the topical report within two weeks of this notification from the contracting officer. This report shall be used in lieu of a final report.

DOE/MC/29113--4048
EM report

ICERVS
Phase II Subsystem Design Report

February 4, 1993

Prepared by:		
	David A. Smith	Date
Approved by:		
	John Wagner	Date

REVISION HISTORY

Revision	Date	
A	February 4, 1993	Original Version

Table Of Contents

Table Of Contents (continued)

Table Of Contents (continued)

Table Of Contents (continued)

List Of Figures

List Of Figures (continued)

Figures for 8.0 thru 11.0 yet to come

List Of Tables

Tables for 8.0 thru 11.0 yet to come

GLOSSARY

2D _

Geometric Object: a geometric object that occupies conceptual space of dimensionality 2. Examples of 2D geometric objects include circles, rectangles, and polygons. A 2D geometric object lies in a plane, occupies surface area, but does not occupy volume.

Polygon: a geometric object characterized by a sequence of connected vertices lying in a plane. The straight line connections between adjacent vertices are called edges. A 2D polygon is *convex* if a line segment joining any two interior points is completely contained within the polygon.

2.5 D Surface Map: projection of a 3D space onto a user specified plane (generally the viewing plane) and is typically represented by a two-dimensional data array, where the two array indices are used to represent two of the dimensional coordinates and the data values are used to represent the third coordinate. The map has a user selectable, uniformly spaced grid.

3 D _

Geometric Object: a geometric object that occupies conceptual space of dimensionality 3. Examples of 3D geometric objects include spheres, rectangular parallelepipeds, and prisms. A 3D geometric object has the property of volume.

Polyhedral Object: a 3D geometric object bounded by plane faces which are polygons.

Swept Volume: the process of creating a 3D geometric object by projecting a 2D geometric object through a path (normal to 2D object) in 3D space.

Category: for a geometric object, the identifier that describes the general type or class for a geometric object, such as, wall, door, barrel, etc.

CCV: closed circuit video. Used to remotely view a location.

Class, C++ _

Base Class: a class which is included in another class(es). A base class typically provides general-purpose characteristics that are tailored by the derived class(es).

Class: a C++ data type defined by the programmer, that aggregates programmer-defined data structures (or member data), member functions, and custom operators.

Derived Class: a class defined such that it includes the member data and member functions from another class. The derived class is said to inherit the characteristics of the other class.

GLOSSARY (continued)

Member Data: the variables and data structures defined within a class (including those inherited from other classes).

Member Function: the member data manipulation functions and other functions for a class (including those inherited from another class).

Client-Server: a style of computer program implementation whereby one program (the server) is configured as a provider of a particular set of services and another program (the client) is a user of those services. The client sends commands to the server which executes the command and returns a result. In a networked environment, clients and servers may reside on different computer systems.

Common Interface for Sensors: A high-level, standard format for communicating sensor commands, data, and user interface information between intelligent sensor subsystems and an application program. The common interface provides a flexible capability for using different sensor configurations in remediation systems.

CRADA: Cooperative Research And Development Agreement signed to transfer technology from a national laboratory to a commercial firm..

CSC: Computer Software Component.

CSCI: Computer Software Configuration Item

CSU: Computer Software Unit

Cut Plane: a plane used to segment 3D space into two regions. Typically one or more cut planes are used to bound a region of interest. A pair of cut planes with a user-setable separation that is used to view slices of the 3D volume is called a slice cut plane.

Derived Property: a property created by the Boolean combination of two or more other properties.

Dialog Box: an area on the user display for user input/output.

DOE: Department Of Energy

GENISAS: General Interface for Supervisor and Subsystems. A software package, developed by Sandia National Laboratory, that provides a set of general communications tools. GENISAS implements a client-server communications model using RPC and UNIX sockets..

GUI: Graphical User Interface.

GLOSSARY (continued)

GISC: Generic Intelligent System Control.

ICERVS: Interactive Computer-Enhanced Remote Viewing System

IGES: Initial Graphics Exchange Standard

IGRIP: A set of software tools developed by Deneb Robotics, Incorporated to facilitate the off-line programming of robotic systems. The tools provide 3D simulation to prototype and visualize robot actions.

INEL: Idaho National Engineering Laboratory

ISOE: Intelligent System Operating Environment. A software package for UNIX low-level, socket-based communications developed by Sandia National Laboratories as part of the GENISAS package.

Level: for a node in a tree, the number of antecedent nodes in the direct path from the node to the root node.

Model
Geometric Model: a computer representation of a physical object.

World Model: a computer representation, used by robotic systems, of a taskspace and/or workspace.

Motif: An XWindows based computer windowing system from Open Systems Foundation. Motif is a standard part of most UNIX based workstations.

Node: one data element within an octree data structure

Child Node: a direct descendant of another node.

Leaf Node: a node with no children. Leaf nodes of an octree have empty, full, or unknown states.

Node State: for octrees, a flag used to describe the occupancy of the region of space corresponding to a node. The four node states are:

Empty	none of the region is occupied
Partial	some (but not all) of the region is occupied. The occupancy is described in further detail by children nodes

Full	all of the region is occupied
Unknown	the occupancy of the region is not known. Initially, all nodes are set to the unknown state.

Object_
Category: the identifier that describes the general type or class for a geometric object, such as, wall, door, barrel, etc.

Geometric Object: a representation of a contiguous region of conceptual space defined in a mathematical form(s). Geometric objects can be two dimensional (e.g. circle, rectangle, and polygon) or three dimensional (e.g. sphere, cube, and prism).

Group: an aggregation of two or more contiguous geometric objects that are treated as a single geometric object.

Octree: an 8-ary tree data structure that represents the spatial occupancy of a 3-dimensional region. The data structure is produced by the recursive subdivision of a finite cubical universe.

OOA: object oriented analysis.

OOD: object-oriented design.

OOP: object-oriented programming.

OOPL: object-oriented programming language, such as C++ and SmallTalk.

ORNL: Oak Ridge National Laboratory

Polygon: a geometric object characterized by a sequence of connected vertices lying in a plane. The straight line connections between adjacent vertices are called edges. A 2D polygon is *convex* if a line segment joining any two interior points is completely contained within the polygon.

Polyhedron: a 3D geometric object bounded by plane faces which are polygons

PNL: Pacific Northwest Laboratory

Primitive/Geometric Primitive: a 3D geometric object typically defined by analytic description (equations). Examples of primitives include spheres, cylinders, and planes (half-spaces).

Prismoid: a polyhedron that has all of its vertices in two parallel planes, and with the same number of vertices in each plane.

Property Data: for octrees, data used to describe the non-geometric physical characteristics of a region in space corresponding to one node. Property data is generally a single value (temperature, conductivity, weight, color, etc.) or a simple array of values (magnitude and phase, speed vector, etc.).

Quadtree: a 4-ary tree data structure that represents the spatial occupancy of a 2-dimensional region. The data structure is produced by the recursive subdivision of a finite square universe.

Region: part of a 2D or 3D space that is of interest at a particular point in time.

Remediation: the process of environmental restoration of hazzardous sites, such as the waste sites found on DOE reservations.

Robotic System: A computer-controlled mechanical mechanism. Typically, the computer coordinates the actions of multiple linked mechanical elements to achieve a programmed path at the working end of the mechanism.

Rogue Wave Canvas: a software package that provides drawing management for 2D and 3D geometric figures on a XWindows screen.

ROI: Region Of Interest.

Rotation: for computer graphic display, an angular movement of the image about one or more coordinate axes of the computer model of a 3D space.

Scaling: for computer graphic display, a change in the size of the image of a computer-modeled space. For ICERVS, scale is described as a percentage of full scale (At full scale, the entire computer-modeled region is made to exactly fit in the display window.)

Sculpting: for octrees, the process of changing node states to empty along a selected path or within a selected region. Sculpting is typically used to clear regions corresponding to input data provided by line-of-sight sensors.

Sensor -
Laser Range Finder: A non-contact sensor using laser energy to measure range to objects in a scene, which may be one to 20 meters from the sensor.

GLOSSARY (continued)

MiniLab: A computer-based system developed by Sandia National Laboratory for modularly configuring, acquiring data from, and integrating data from a suite of sensors. MiniLab provides a set of common system interfaces and a flexible user interface.

Structured Light: A non-contact, dimensional profiling system that use triangulation between a laser source, projected onto a region of interest, and an imaging detector such as a video camera.

Sensor: a device for measuring some physical quantity, such as temperature, weight, pressure, etc.

Sensor Data: the location, orientation and other attributes of a sensor together with a response set. Typically, sensor data is stored at the node representing the sensor location. One or more kinds of property data can generally be derived from this sensor data.

Sensor Subsystem: a subsystem that contains or simulates at least one sensor and outputs sensor data to another system. A sensor subsystem typically contains a local computer for control and computation. The software on the sensor computing platform is generally configured as a server. Client applications on other computing platforms connect to the sensor subsystem software in order to control the sensor and obtain sensor data.

Site: a particular waste location and its environs. Examples would be an underground storage tank, a buried pit, a production facility, etc.

Slice Cutplane: a pair of cut planes with a user defined separation used to view slices of the 3D volume. The pair of cut planes are controlled as a single object.

SNL: Sandia National Laboratory

Spatial Data: a data base containing discrete elements with dimensionality of three (solids). The data base is used to describe a 3D modeling space.

STEP: Standard for The Exchange of Product model data (ISO 10303). A computer-based method for specifying and exchanging information in engineering and manufacturing systems. STEP defines product specifications, such as geometry/shape, material, tolerances, behavior, function, product structure, and process sequences.

Taskspace: A 3D region of space that is of interest, typically a portion of a workspace. Taskspace and workspace are often used interchangebly.

Translation: for computer graphic display, a change in the apparent position of the image of a computer-modeled space. Translation, when used in conjunction with scaling, provides a means for viewing a subsection of the world model at a magnified scale. For ICERVS, translation is described as screen-horizontal and screen-vertical displacements in external units.

Tree: a data structure characterized by a hierarchy of elements or nodes descendant from a single or root node.

Tree _
Tree Traversal: the process of retrieving each node from a tree in a prescribed order. Typically each node is retrieved one time. A *depth-first* tree traversal retrieves the descendants of a node before retrieving the siblings of a node.

TrueSolid: A software package from Octree Corporation that provides tools for the building, viewing, modification of volumetric data sets using octrees.

Units _
External Units: the physical dimensional units selected by the ICERVS user for display and data input/output.

Internal Units: the physical dimensional units used internally by ICERVS. For ICERVS, these units are SI (dimensions in meters).

Tree Units: the dimensional units used by the octree software to describe octree space. For ICERVS these dimensions vary between zero and 2,097,152 (2^{21}). This also sets the maximum spatial resolution of the octree.

UNIRAS Toolmaster: a collect of software products that facilitate development of graphics applications. A 2D and 3D graphics software library (agX) and a GUI builder utility (UIM/X) that facilitates the definition and implementation of GUI components are included in Toolmaster.

UNIX: The computer operating system used on most major workstations. UNIX is a multi-user, multi-tasking computing environment.

Volumetric Data: a data base containing discrete elements with spatial dimensionality of three (solids). The data base is used to describe a 3D modeling space.

VxWorks: A real-time, multi-tasking operating environment for VME-based computers.

GLOSSARY (continued)

Wire frame: a method for defining or generating a graphic display of a polyhedron as a set of points and connecting edges.

Workspace: A 3D region of space that is of interest. A workspace is often subdivided into several taskspaces. In many cases, taskspace and workspace are used interchangebly.

1.0 INTRODUCTION

1.1. Scope

This ICERVS Phase II Subsystem Design Report describes the detailed software design of the Phase II Interactive Computer-Enhanced Remote Viewing System (ICERVS).

ICERVS is a computer-based system that provides data acquisition, data visualization, data analysis, and model synthesis to support robotic remediation of hazardous environments. Because of the risks associated with hazardous environments, remediation must be conducted remotely using robotic systems, which, in turn, must rely on 3D models of their workspace to support both task and path planning with collision avoidance. Tools such as ICERVS are vital to accomplish remediation tasks in a safe, efficient manner.

The 3D models used by robotic systems are based on solid modeling methods, in which objects are represented by enclosing surfaces (polygons, quadric surfaces, patches, etc.) or collections of primitive solids (cubes, cylinders, etc.). In general, these 3D models must be created and/or verified by actual measurements made in the robotics workspace. However, measurement data is empirical in nature, with typical output being a collection of xyz triplets that represent sample points on some surface(s) in the workspace. As such, empirical data cannot be readily analyzed in terms of geometric representations used in robotic workspace models. The primary objective of ICERVS is to provide a reliable description of a workspace based on dimensional measurement data and to convert that description into 3D models that can be used by robotic systems. ICERVS will thus serve as a critical factor to allow robotic remediation tasks to be performed more effectively (faster, safer) and economically than with present systems.

1.2. Document Overview

Section 1.0 provides an introduction to this document including a definition of scope and an overview of the contents of the document.

Section 2.0 provides a list of related documents that provide ancillary information that may be useful in understanding this document.

Section 3.0 provides a brief background of the ICERVS system including summaries of the implementation phases, mission profiles, system requirements, and the success criteria for Phase II.

Section 4.0 summarizes the Phase II architecture as defined of the ICERVS Phase II System Design Report (October 31, 1993).

Section 5.0 details the client-server communications design. Command and message protocols are defined; message formats are detailed; software classes are identified; and major function descriptions are given.

THIS PAGE INTENTIONALLY LEFT BLANK

2. RELATED DOCUMENTS

Phase I MTI technical proposal Q2-030, "*Interactive Computer Enhanced Remote Viewing System*", December 13, 1991.

Phase II MTI technical proposal Q3-419, "*Interactive Computer Enhanced Remote Viewing System*", May 14, 1993.

MTI ICERVS "*Software Development Plan*", December 23, 1992.

MTI "*ICERVS Phase I System Design Report*", December 21, 1992.

MTI "ICERVS Phase I Subsystem Design Report - Phase 1", April 27, 1993

MTI 98TR25 "ICERVS Phase 1 Topical Report", April 27, 1993

MTI "*ICERVS Phase II System Design Report*", November 15, 1993.

THIS PAGE INTENTIONALLY LEFT BLANK

3. BACKGROUND

This section provides background information on ICERVS. An overview of the ICERVS system is provided including a discussion of the implementation phases and a review of the mission profiles and system requirements. The final subsection defines the success criteria for Phase II, which provide a framework for the Phase II design and implementation efforts.

3.1. ICERVS Implementation Phases

The development of ICERVS has been structured into three phases based upon the maturity level of its constituent technologies. The features to be developed in each phase are listed in Table 3-1. Each of the ICERVS phases is briefly described below:

The Phase I objective is to achieve Maturity Level III, Subscale Major Subsystems, for those portions of ICERVS that had been demonstrated at the research laboratory scale. The majority of the work in this phase involves the development of the basic analysis software capabilities for octree data storage, sensor data visualization, and geometric object creation and manipulation.

The Phase II objective is to achieve Maturity Level IV, Subscale Integrated System, in which ICERVS will be demonstrated at a limited size to enable successful implementation of a full-scale system in Phase III. The result of this work will be an integrated system that provides a subscale structured light system; an upgraded computing platform and enhanced analysis software to include data analysis and visualization tools; enhanced facilities for creating geometric objects; and an integrated user interface.

The Phase III objective is to achieve Maturity Level V, Full-Scale Demonstration, in which ICERVS will be demonstrated in the intended application. The result of this work will be a full-scale, integrated system for use in the remediation of underground storage tanks. This system will include a full-scale structured light sensor subsystem; an interface to a robot controller; a facility for accepting geometric models derived from architectural data; and enhanced data analysis and visualization tools.

3.2. Mission Profiles

As part of the ICERVS preliminary design, mission profiles were described for three remediation tasks. The original profiles are documented in the ICERVS Phase I System Design Report Section 3. During the initial stages of the Phase II design effort, it became apparent that some minor revisions in the mission profiles were necessary in order to clarify points or to reflect the greater insight into the needs of the DOE community. The details of the revisions to the mission profiles can be found in Appendix A of the ICERVS Phase II System Design Report. Appendix B of that document contains an updated copy of the Mission Profiles, including the revisions.

Table 3-1 Phased Implementation Of ICERVS Features

Analysis Software	Computing Platform	Sensor Subsystems
Phase I Features		
• Volumetric Data - Tree construction and manipulation - Tree structure display - Tree storage display - Tree utility programs • Geometric object data - Object definition - Associated text • Display - Orthogonal projection - Translation - Scaling - Cut planes along major axes - Multiple windows - Pseudo-color display • Model Building - Polygon generator - Region of interest - 3D projection (prismatic models)	• Silicon Graphic workstation (Indigo) • Color monitor • GL graphics library	• Simulated data
Additional Phase II Features		
• Volumetric Data - Interface for real empirical data - Storage, display and analysis of material property data • Geometric object data - Output in IGES format • Display - Arbitrary point of view - Arbitrary cut planes - Surface shading - Synchronized video and computer graphics images • Model Building - Arbitrary polyhedron - Sculpting of octree model - Quantitative comparison between volumetric data and geometric object	• Real-time CCTV • Operator control (track ball) for interactive viewing • Control interface to sensor subsystem • Upgraded CPU with additional internal memory	• Two subscale instrument stations with - Solid state cameras - Laser illuminators - Positioning in pitch • Sensor controller (PC) - Image analysis - Surface profiling - Automatic mapping - Interface to computing platform • Simulated waste tank
Additional Phase III Features		
• Geometric object data - Storage of architectural plans - Input with IGES - Ability to edit IGRIP models • Display with perspective view • Model Building - Automatic modeling of simple surfaces - Surface connectivity tool - Surface model templates	• Link to robotic controller (such as COPILOT)	• Three full-scale instrument stations • Color camera and grid projector • Automatic positioning in azimuth • Absolute tank coordinate frame established by surveying tools • Improvement suggested by Phase II • Design of hardened station

3.3. System Requirements

From the mission profiles, a list of ICERVS primary requirements were developed. These requirements are documented in the ICERVS Phase I System Design Report, Section 3. In the course of detailing the Phase I subsystem design, it became apparent that some minor revisions in the requirements were necessary in order to clarify points or to reflect the greater insight into the roles of the various subsystem elements. The details of the Phase I revisions to the requirements can be found in Appendix A of the ICERVS Phase I Subsystem Design Report.

During the Phase II design, revisions were made once again to the requirements to reflect the additional insights gained from the Phase I experience. In addition, the modification of the Mission Profiles (paragraph 3.2) and the inclusion of the buried waste task in Phase II has necessitated the addition of several new system requirements. (Table 3-1 was also modified to reflect the changes in the system requirements.) The details of these revisions to the requirements can be found in Appendix C of the ICERVS Phase II System Design Report. Appendix D of that document contains an updated copy of the ICERVS System Requirements, including all revisions. The ICERVS system requirements are summarized in Table 3-2 below. Annotations are included with the table to indicate the applicable phase for each requirement.

3.4. Phase II Success Criteria

This program aspires to make ICERVS the de facto standard for volumetric data storage, manipulation and presentation in environmental restoration systems. Toward that goal, the ICERVS Phase II system will be successful if:

1 The sensor subsystem automatically and/or interactively maps the surface and walls of the simulated single-shell tank.

2 The ICERVS properly creates a volumetric data base for the simulated single-shell tank and maintains an equivalent world model in a form compatible with the needs of robotic controllers.

3 The ICERVS demonstrates enhanced viewing and model-building tools.

4 The ICERVS displays corresponding (matching) scenes from live CCV and the computer's world model as the operator pans the camera.

The specific success criteria for the buried waste material properties task with INEL are:

5 Accept, align, store, and display material property data from two or more sensors that have scanned a selected segment of the workspace.

6 Fill in missing material data points through interpolation.

7 Combine two sets of property data by Boolean means to form a third (derived) property, store, and display it in an appropriate manner.

8 Update the volumetric and geometric object data, as needed, in response to partial excavation.

Table 3-3 relates the success criteria and the system requirements. In some cases, a single requirement may be supportive of several success criteria, in which case the requirement will be listed multiple times.

Table 3-2 Summary Of ICERVS Requirements

NO	SYSTEM REQUIREMENT	PHASE	NOTES
R1	**DATA REPRESENTATION**		
R1.01	Octree: spatial data	1	
R1.02	Octree: property data	3	partial for phase 2
R1.03	Octree: spatial interpolation	2	
R1.04	Octree: linear resolution 1:512, expandable	1	
R1.05	Geom: polyhedral objects	1	
R1.06	Geom: geometric primitives	2	
R1.07	Geom: associated text each object	1	
R1.08	Geom: 100 objects, expandable	1	
R1.09	Geom: enter architectural and robot plans	3	design influence
R1.10	Octree: sensor data	3	partial for phase 2
R1.11	Octree: property data interpolation	2	
R2	**OBJECT MODELING**		
R2.01	Library of primitives / templates	2	
R2.02	Standard templates	2	
R2.03	User-defined templates	3	partial for phase 2
R2.04	Automatic waste surface modeling	3	design influence
R2.05	Synthesize 2D polygons	1	
R2.06	Synthesize 3D polyhedra	2	
R2.07	Dimensioning tools	2	
R2.08	Attach text to geometric objects	1	
R3	**COMPUTER GRAPHICS DISPLAY**		
R3.01	Translation and scaling	1	
R3.02	Display coordinate axes	2	
R3.03	Parallel cut planes	2	
R3.04	Display geometric object text data	1	
R3.05	Shaded or wire frame geometric object display	2	
R3.06	Update display as points received	1	
R3.07	Pseudo-color octree data	2	
R3.08	Color geometric objects by category	2	
R3.09	Text display view parameters	2	
R3.10	Save / Recall view parameter set	2	
R3.11	Multiple windows displaying same data	1	
R3.12	View tracks sensor station attitude	2	
R3.13	Display 2.5D surface map	2	completed in phase 1
R3.14	Display views of spatial and property data	3	phase 1,2: orthographic phase 2: orthogonal & arbitrary

Table 3-2 Summary Of ICERVS Requirements (cont.)

NO	SYSTEM REQUIREMENT	PHASE	NOTES
R4	VIDEO DISPLAY		
R4.01	Monitor for each camera plus one for processing	3	partial for Phase 2
R4.02	Display wire frame geometric objects over video	3	design influence
R4.03	Real-time CCV	3	phase 2: camera on sensor
R4.04	Flexible color TV camera	3	
R5	MANIPULATION AND ANALYSIS		
R5.01	Copy octree	2	
R5.02	Set region within octree to selected state	2	
R5.03	Operator delete geometric objects	2	
R5.04	Scan object for consistency with octree	2	
R5.05	Compare octree and object data	2	
R5.06	Compare two octrees, compute difference	2	
R5.07	Compute 2.5D surface map from octree	2	
R5.08	Compute difference 2.5D surface maps	2	
R5.09	Surface connectivity	3	design influence
R5.10	Tools for registering data from different sensors	3	partial for phase 2
R5.11	Combine property data via Boolean operations	2	
R6	MISCELLANEOUS FUNCTIONS		
R6.01	Edit system parameters	2	
R6.02	Save / Retrieve waste site data sets to/from disk	1	
R6.03	Build octree from backup raw data	2	
R6.04	Maintain operator log	2	
R6.05	Multiple system of units	2	
R6.06	Define disassembly data	3	design influence
R6.07	Establish Application Data Structure	2	
R6.08	Support multiple active datasets per waste site	2	
R6.09	Support multiple active waste sites	2	
R7	DATA INTERFACE		
R7.01	Input: x,y,z position	2	
R7.02	Input: optional resolution	2	
R7.03	Input: optional property value	2	
R7.04	Input: optional sensor location	2	
R7.05	Input: station angles during visual inspection	2	
R7.06	Output: geometric model data	2	
R7.07	Output: 2.5D surface map data	2	
R7.08	Input: sensor data	3	phase 2: simulate laser range finder and MiniLab

Table 3-2 Summary Of ICERVS Requirements (cont.)

NO	SYSTEM REQUIREMENT	PHASE	NOTES
R8	OPERATOR INTERFACE		
R8.01	Graphic tools	2	
R8.02	Provide operator help function	3	design influence
R8.03	Provide hard copy output	3	design influence
R9	SENSORS		
R9.01	Tele-operate position and rate commands	2	
R9.02	Tele-operation display line of sight	3	partial for phase 2
R9.03	Tele-operation text display station angles	3	partial for phase 2
R9.04	Automatically map surfaces	2	
R9.05	Operator parameters	2	
R9.06	Draw / display scan paths	2	
R9.07	Continual backup of raw data	2	
R9.08	not used		
R9.09	Operate sensor remotely from computing platform	2	
R9.10	Surface mapping sensor	3	partial for phase 2
R9.11	Sensor performance	3	partial for phase 2
R10	SITE ENVIRONMENT		
R10.01	Surface characteristics	3	design influence
R10.02	Illumination and visibility	3	design influence
R10.03	Environmental considerations	3	design influence
R10.04	Design constraints	3	design influence

Table 3-3 Allocation Of System Requirements To Phase II Success Criteria

No.	Success Criteria	Full Implementation	Partial Implementation
1	Automatically or interactively maps surface and walls of simulated tank	R9.01, R9.04, R9.05, R9.06, R9.07, R9.09	R9.02, R9.03, R9.10, R9.11
2	Properly creates volumetric data base and maintains equivalent world model.	R1.01, R1.03, R1.04, R1.05, R1.06, R1.07, R1.08, R5.01, R5.03, R5.04, R5.05, R5.06, R5.07, R5.08, R6.01, R6.02, R6.03, R6.04, R6.05, R6.07, R6.08, R6.09, R7.01, R7.02, R7.06, R7.07	
3	Demonstrates enhanced 1. model building tools and 2 viewing tools	 R2.01, R2.02, R2.05, R2.06, R2.07, R2.08, R3.04, R3.05, R3.08 R3.01, R3.02, R3.03, R3.06, R3.07, R3.09, R3.10, R3.11, R3.12, R3.13, R8.01	 R2.03 R3.14
4	Displays corresponding scenes from live CCV and world model.	R3.12, R7.04, R7.05, R9.01, R9.09	R4.01, R4.03, R9.02, R9.03
5	Accept, align, store, and display sensor property data	R7.03, R9.09	R1.02, R1.10, R3.14, R5.10, R7.08
6	Interpolate sparse material property data	R1.11	
7	Combine two sets of property data by Boolean means.	R5.11	R3.14
8	Update the geometric data in response to partial excavation.	R5.01, R5.02, R5.04, R5.06	

4. PHASE II ARCHITECTURE

This section summarizes the ICERVS Phase II architecture in preparation for the presentation of the detailed design information in subsequent sections of this document. For a more complete description of the ICERVS architecture, refer to the ICERVS Phase II System Design Report.

4.1. System Architecture

In general, an ICERVS will consist of a computing platform, one or more sensor subsystems for data acquisition, one or more CCV subsystems, an interface to a robotic controller/simulator, and analysis software for data processing. The system outputs include world model information, CAD models, 2.5D maps, and data for archiving. Figure 4-1 summarizes the ICERVS system and illustrates its functional model. Figure 4-2 shows the ICERVS architecture that will be demonstrated as part of Phase II. The following paragraphs discuss the Phase II architecture.

4.1.1. Computing Platform

The hardware kernel of ICERVS is a Silicon Graphics (SGI) Indigo workstation. The Phase I Indigo configuration included a RISC 3000 CPU, 16 Mbyte memory, 400 Mbyte hard disk, 19" color monitor, and trackball. For Phase II, the Indigo is being upgraded to incorporate a RISC 4000 CPU, an additional 16 Mbyte memory (32 Mbyte total), and an additional 540 Mbyte hard disk (940 Mbyte total). SGI system software upgrades include UNIRAS ToolMaster, Rogue Wave C++ library updates, SNL ISOE (Intelligent System Operating Environment), Octree Corporation's TrueSolid software, and several software development tools (debuggers, editors, etc.).

A Sun SPARC Classic workstation has been added to augment the ICERVS SGI computing platform. The Sun workstation is configured with a SPARC CPU, 16 Mbyte memory, 424 Mbyte hard disk, and a 15" color monitor. Major system software components include UNIX, Open Windows and Motif window managers, SNL ISOE, and the ORNL structured light standalone control code. The main functions of the Sun for the ICERVS Phase II effort are (1) to provide additional software development resources to facilitate the implementation of Phase II; (2) to provide a networked environment to test ICERVS; and (3) support the ICERVS interface with the structured light sensor subsystem, the laser range finder simulator, and the MiniLab simulator.

4.1.2. Sensor Subsystems

The data acquisition part of ICERVS Phase II is configurable in order to meet the needs of the specific application. Each sensor subsystem will typically include a local computer for control and computation. As such, each sensor subsystem can be operated as a stand-alone system if desired. However, key to each sensor subsystem is the capability of operating remotely from an integrated user interface resident on the ICERVS computing platform. To support different sensor configurations, the ICERVS system design includes a Common Interface For Sensors for communications between the computing platform and the sensor subsystems. This Common Interface For Sensors will establish a high-level, standard format for communicating sensor commands, sensor data, and screen display information. Each sensor subsystem will define and implement their individual commands, data, and screen displays to be consistent with the standard.

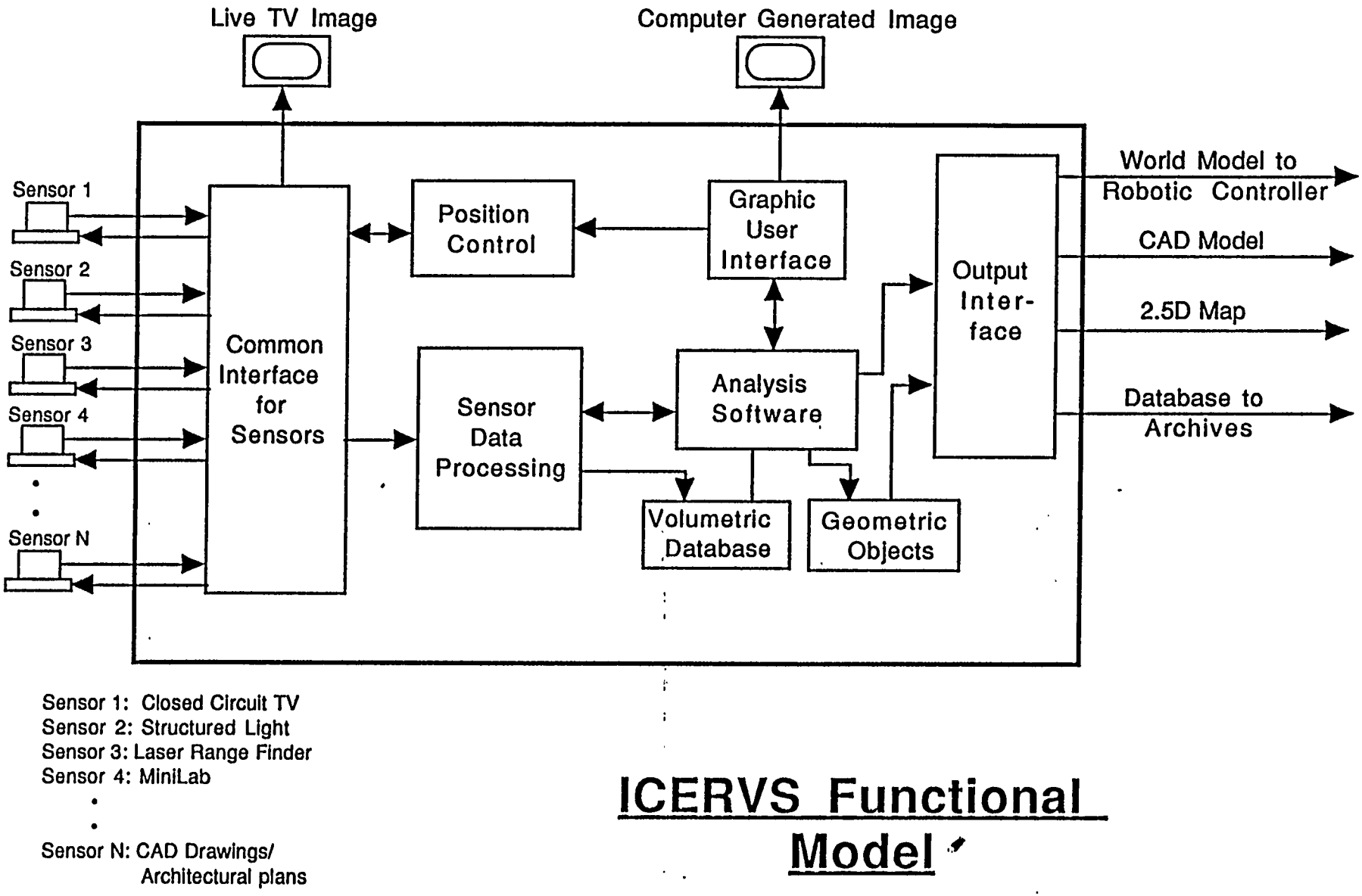


Figure 4-1 -- ICERVS General System Architecture

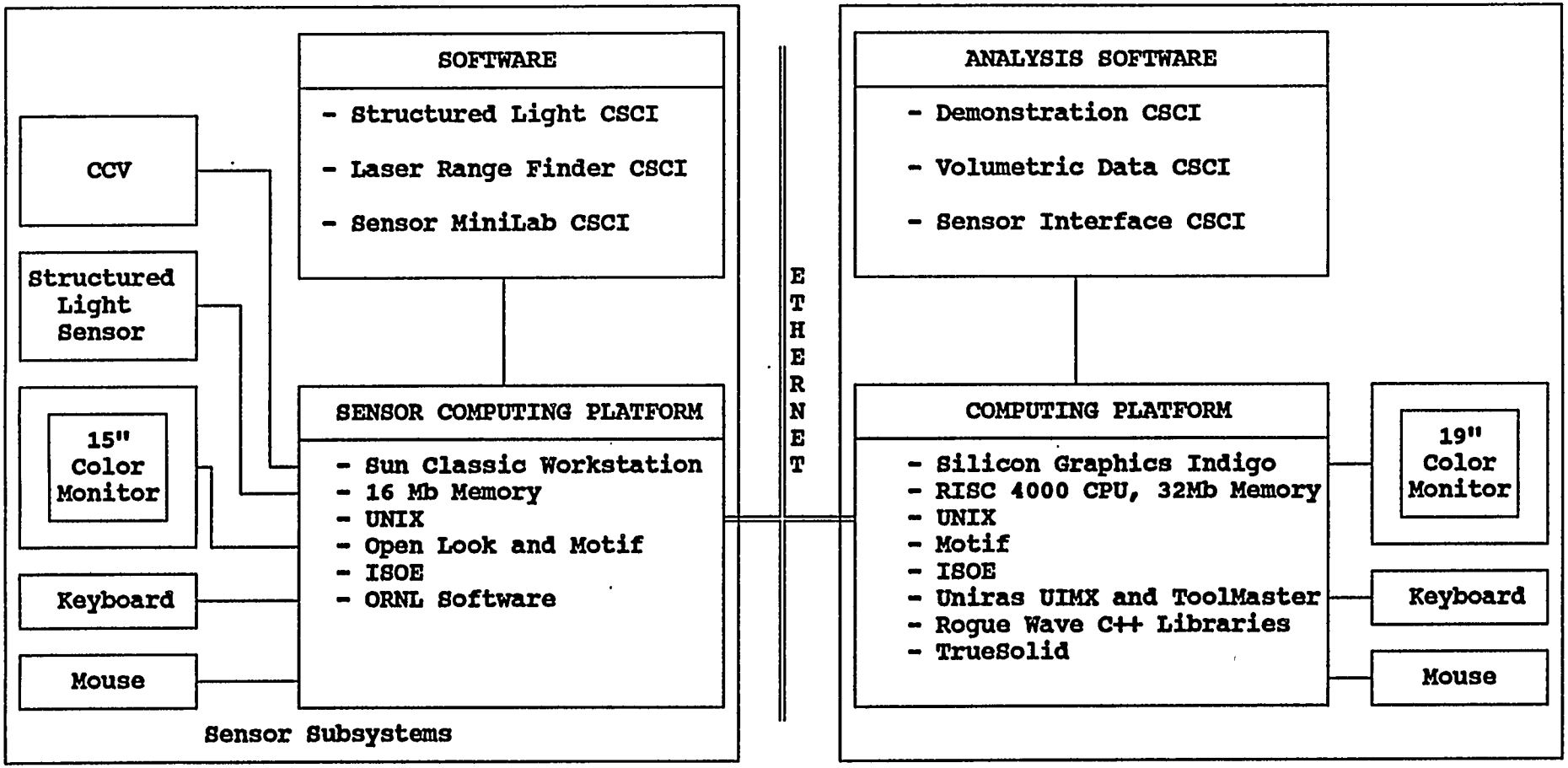


Figure 4-2 ICERVS Phase II System Architecture

For ICERVS Phase II, the three sensor subsystems (Structured Light Sensor, Laser Range Finder Simulator, and MiniLab Simulator) will all be implemented on the Sun Classic platform. The software for each sensor subsystem will act as a server and will interface to the SGI-based computing platform analysis software package.

4.2. Analysis Software Architecture

The ICERVS analysis software runs on the computing platform, interfaces with one or more sensor subsystems, converts sensor data into a common data structure, and stores such data from multiple sensor subsystems. The data from the sensors can be stored together with spatial information to generate a volumetric description of the workspace. The analysis software can provide a visual display of the volumetric and property data for a workspace from arbitrary viewpoints, with numerous tools for modifying, selecting, or highlighting the data according to it characteristics and/or regions of interest. These capabilities allow an operator to easily analyze data to obtain maximum insight prior to and during robotic operations.

Also included in the ICERVS analysis software is a geometric modeling capability that enables the operator to interactively define, manipulate, and output geometric objects that represent features of interest in the workspace. By defining models for these features and passing them to the robotic system, the remediation tasks can be more readily accomplished with an accurate 3D description of the "as-is" condition of the workspace.

The design and implementation of ICERVS software employs an object-oriented approach, which has three distinct parts: object-oriented analysis (OOA), object-oriented design (OOD), and object-oriented programming (OOP).

OOA is concerned with capturing the knowledge about the Problem Domain Component (PDC). For the ICERVS system, the problem domain can be summarized by the following definition:

> ***ICERVS is a computer-based system that provides remote viewing, data acquisition, stored data visualization, data analysis, and model synthesis to support robotic remediation of hazardous environments.***

OOD adds detail to the OOA model by including additional classes for the Human Interface Component (HIC), Data Base Component (DBC), and Task Management Component (TMC).

OOP implements software for each class defined in OOA and OOD. An important consideration is that frequent tangible results should be demonstrated every few weeks.

Figure 4-3 presents the OOA model for the ICERVS Phase II system. It identifies the major system level classes. The diagram follows the Coad-Yourdon method, but uses slightly different symbology. Simple rectangles represent classes (attributes and behaviors are ignored). Various types of lines are used to capture the relationships between classes. A legend is provided on the diagram.

The analysis software architecture for Phase I was fairly simple; it consisted of a single CSCI (IcerMain) which contained three CSCs (Octree Engine, Object Modeling, User Interface). Initially, Phase II was to follow the same architecture. However, it became apparent throughout Phase I and particularly at the Phase I demonstration that ICERVS had a broader application than originally envisioned. It was also apparent that there were several views of what ICERVS could be in addition to its role as input to robotic controllers. Some viewed ICERVS as a database; others saw it as a high level interface to sensors; while others focused on the integration of the database with the sensors. Almost everyone saw ICERVS as part of a solution and one element of an overall remediation system. Therefore, the ICERVS Phase II architecture was refined according to five guiding principles:

1. Consider the needs of both end users and system integrators of remediation systems
2. Develop ICERVS to be consistent with DOE architecture for remediation systems
3. Develop ICERVS to provide complete and general-purpose capabilities
4. Develop ICERVS as a set of independent system components
5. Implement ICERVS components as UNIX client-server programs

The software architecture for ICERVS Phase II is shown in Figure 4-4. The Phase II software will be composed of three primary Computer Software Configuration Items (ICERVS Phase II Demonstration CSCI, Volumetric Data CSCI, and Sensor Interface CSCI) and three sensor subsystem CSCIs (Structured Light Sensor CSCI, Laser Range Finder CSCI, and MiniLab CSCI). The Volumetric Data CSCI and the three sensor subsystem CSCIs act as servers. The ICERVS Phase II Demonstration CSCI acts as a client. The Sensor Interface CSCI acts as both a server and a client. Each CSCI is composed of one or more Computer Software Components (CSCs).

4.2.1. ICERVS Phase II Demonstration CSCI

The ICERVS Phase II Demonstration CSCI is the topmost application level process and contains a high-level operator interface to demonstrate the functions and features of the ICERVS Phase II system. This CSCI is representative of a user application and may be used as a template for creating a user application. A user application will generally be composed of a top level user interface and other software logic that implements the user's particular application. Figure 4-5 shows the ICERVS Phase II top level screen and menu.

Each application must also have an associated Application Data Structure organized and formatted in accordance with ICERVS requirements. The general organization for the Application Data Structure is shown in the following table:

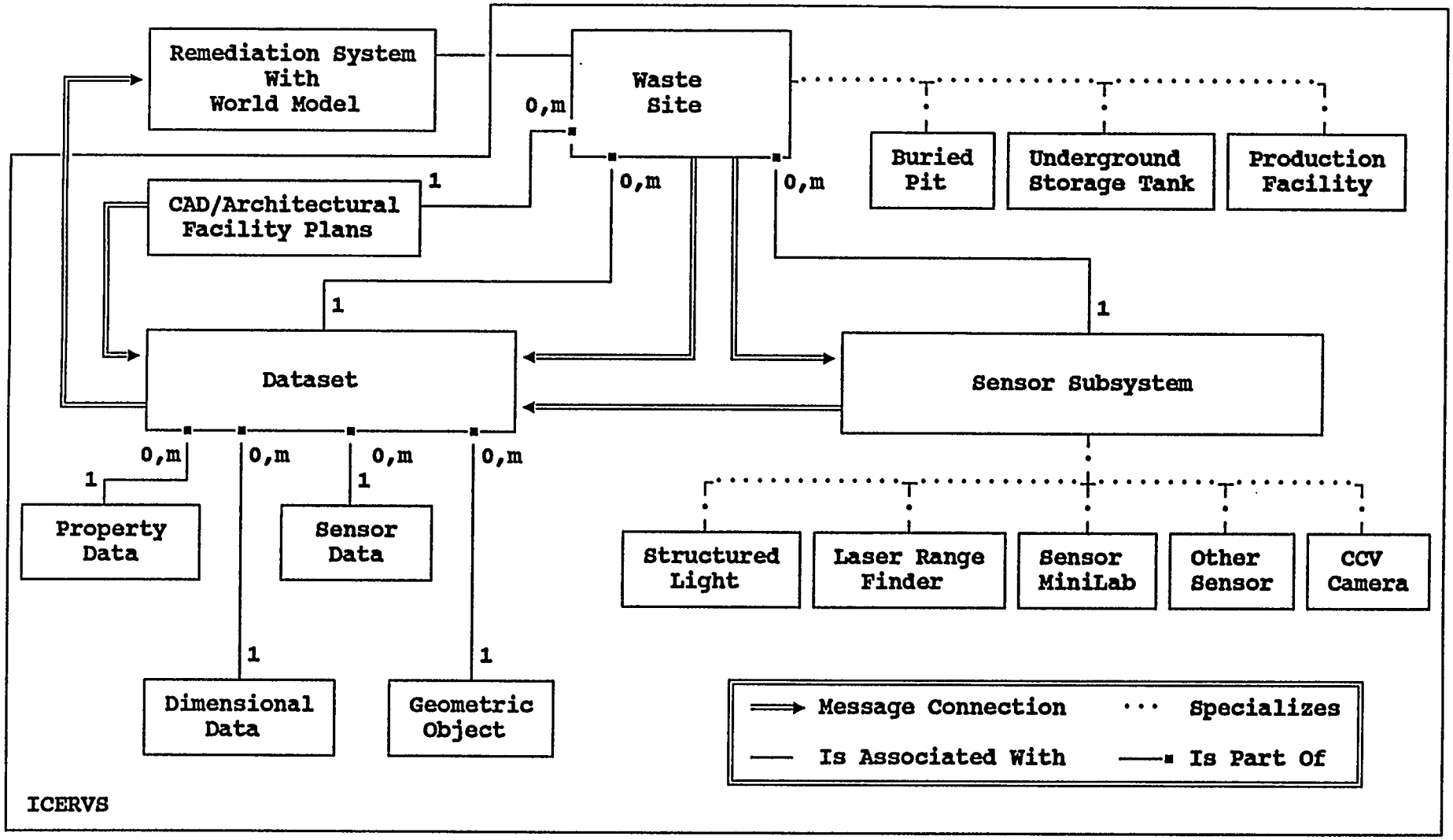


Figure 4-3: ICERVS System Level OOA Diagram

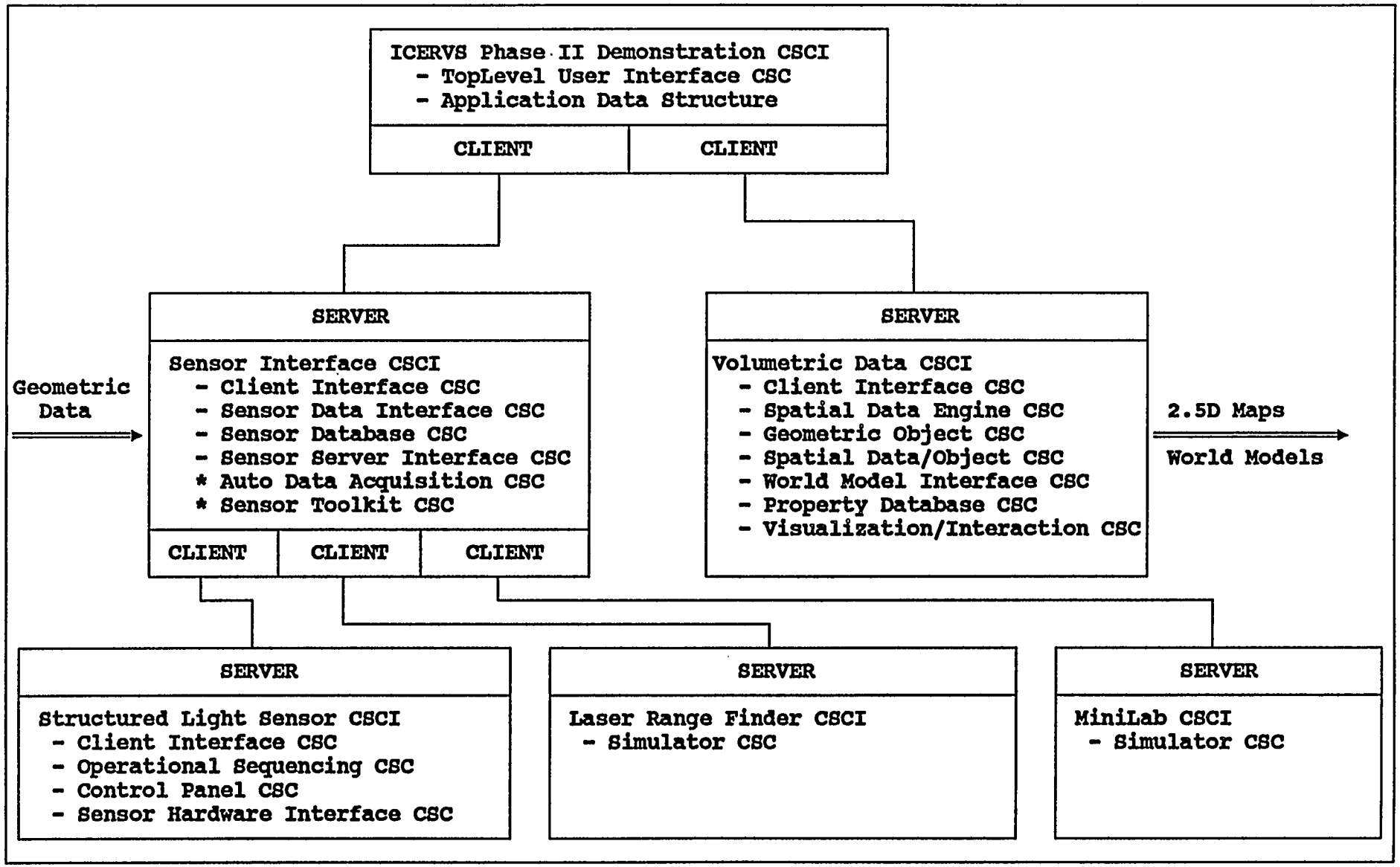


Figure 4-4: ICERVS Analysis Software Architecture

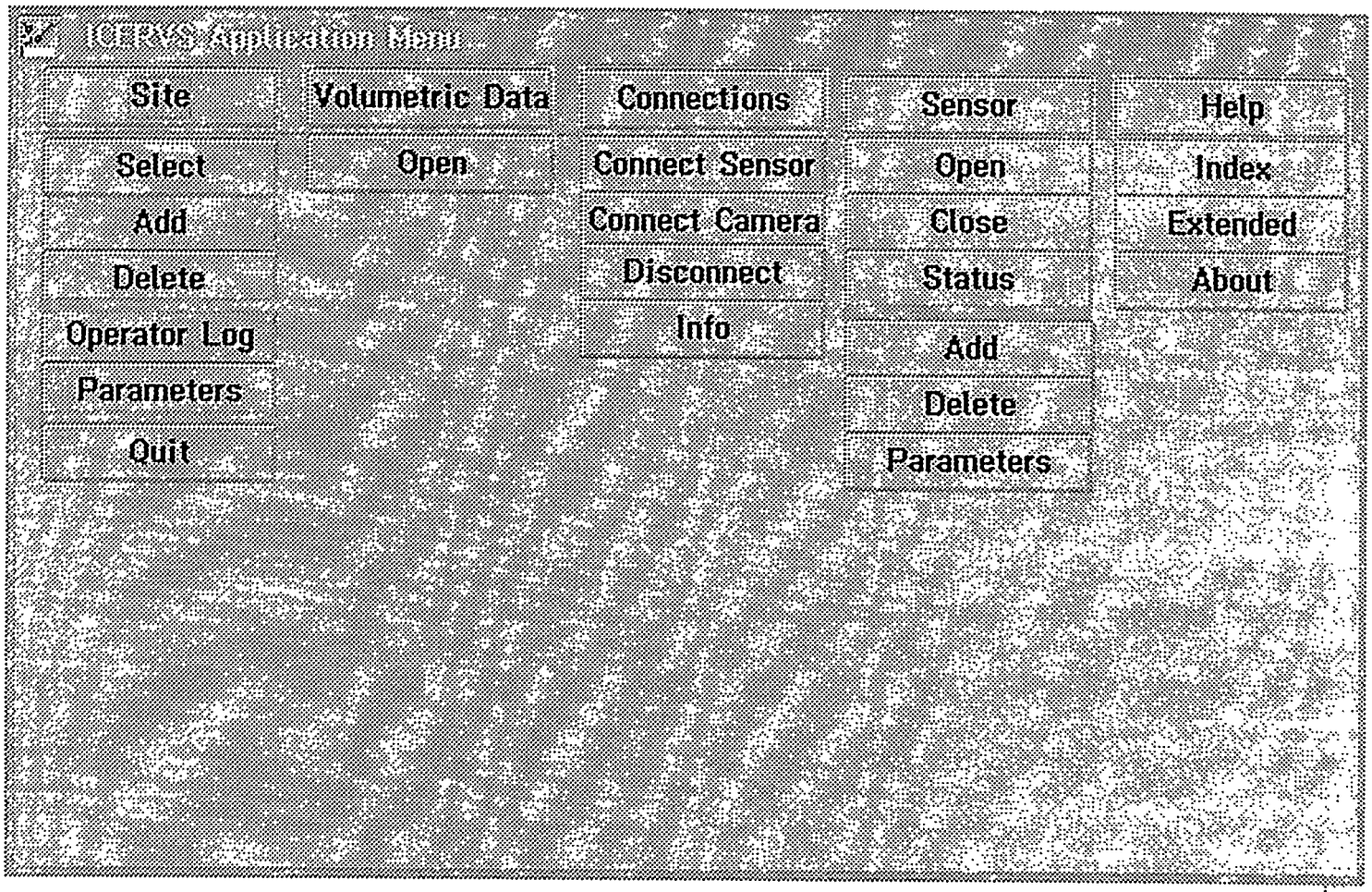


Figure 4-5 -- ICERVS Phase II Top Level Screen and Menu

/usr/SOMEWHERE/...	An arbitrary point in users disk hierarchy
/ICERVS	Root of ICERVS Data Structure Contains template files and default files
/etc	Contains general control files
/SITE1	Contains parameters files for first site, raw data files, and volumetric/geometric data files
/SITE2	Contains parameters files for second site, raw data files, and volumetric/geometric data files
.	
.	

4.2.2. Volumetric Data CSCI

The Volumetric Data CSCI encapsulates the storage, retrieval, manipulation, and visualization of spatial, property, and geometric object data.. It is implemented as a UNIX server. Six of its CSCs (excepting the Visualization/Interaction CSC) form the core of the ICERVS Volumetric Data Server. The Visualization/Interaction CSC will act as client of the ICERVS Volumetric Data Server. Figure 4-6 illustrates the Volumetric Data CSCI and its CSCs. The double-line connections represent the path followed by commands to the server. The single-line connections represent inter-CSC function calls within the CSCI.

4.2.3. Sensor Interface Subsystem CSCI

The Sensor Interface Subsystem CSCI manages the interface to the ICERVS sensor subsystems and provides functions common to all sensor subsystems, such as Add Sensor, Delete Sensor, Connect Sensor, Disconnect Sensor, etc. In the future, this CSCI will provide an automated data acquisition capability and tools for converting the sensor data into a form compatible with the volumetric database. (The Phase II system will implement these capabilities in the Structured Light Sensor CSCI and will require the Laser Range Finder and MiniLab Simulators CSCIs to provide spatial data.). The Sensor Interface CSCI is implemented as a UNIX server. In addition, it is a client of each attached sensor subsystem server. Figure 4-7 illustrates the Sensor Interface CSCI and its CSCs. The double-line connections represent the path followed by commands to the server. The single-line connections represent inter-module function calls within the CSCI

4.2.4 Sensor Subsystem CSCIs

A Sensor Subsystem CSCI manages a sensor subsystem and interfaces to the ICERVS in accordance with the Common Interface for Sensors specification. Each sensor subsystem CSCI is implemented as a UNIX server. Figure 4-8 illustrates a Generalized Sensor Subsystem CSCI and its CSCs. The double-line connections represent the path followed by commands to the server. The single-line connections represent inter-module function calls within the CSCI. The Generalized Sensor Subsystem CSCI is conceptual and

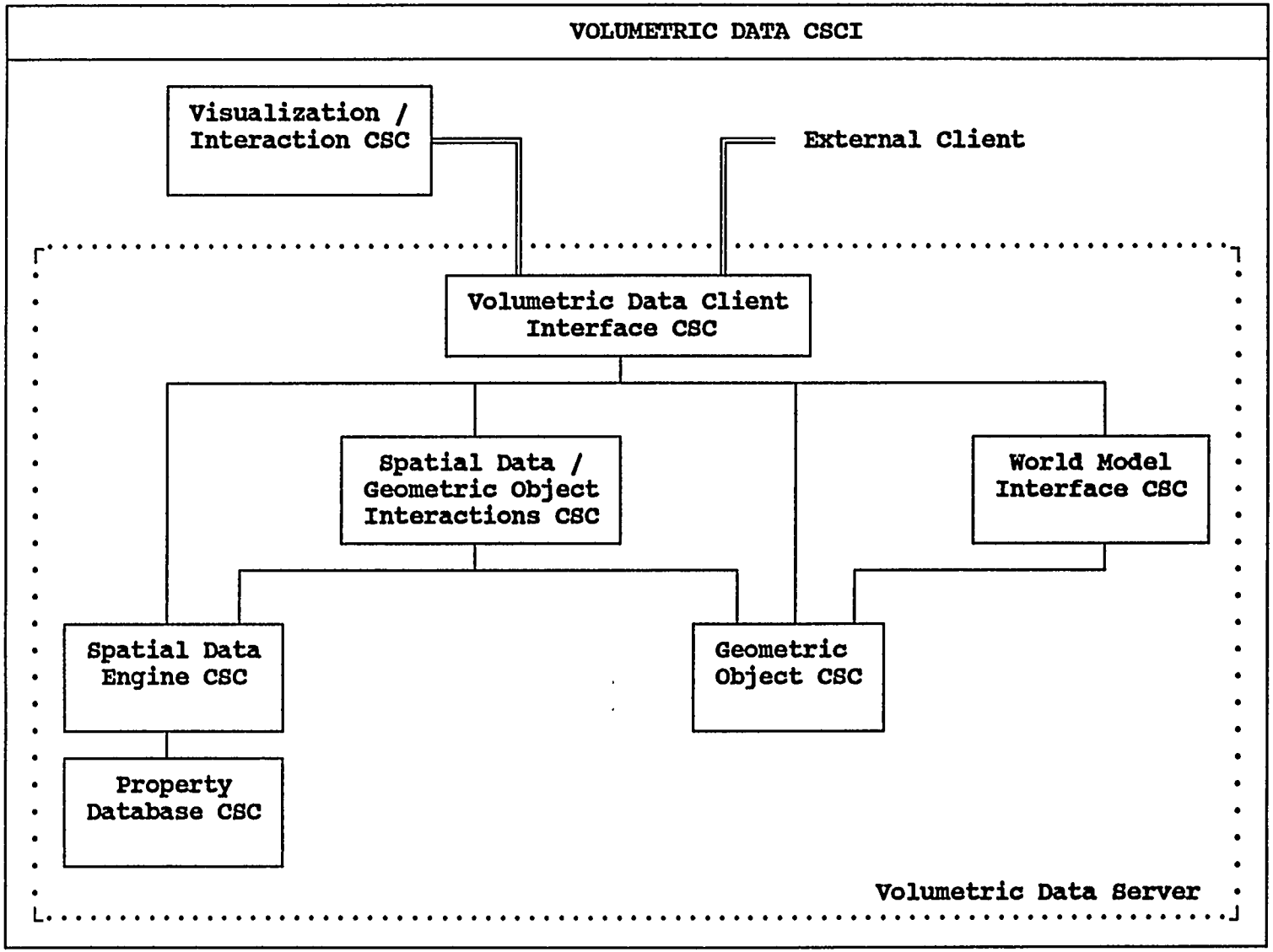


Figure 4-6: Volumetric Data CSCI Block Diagram

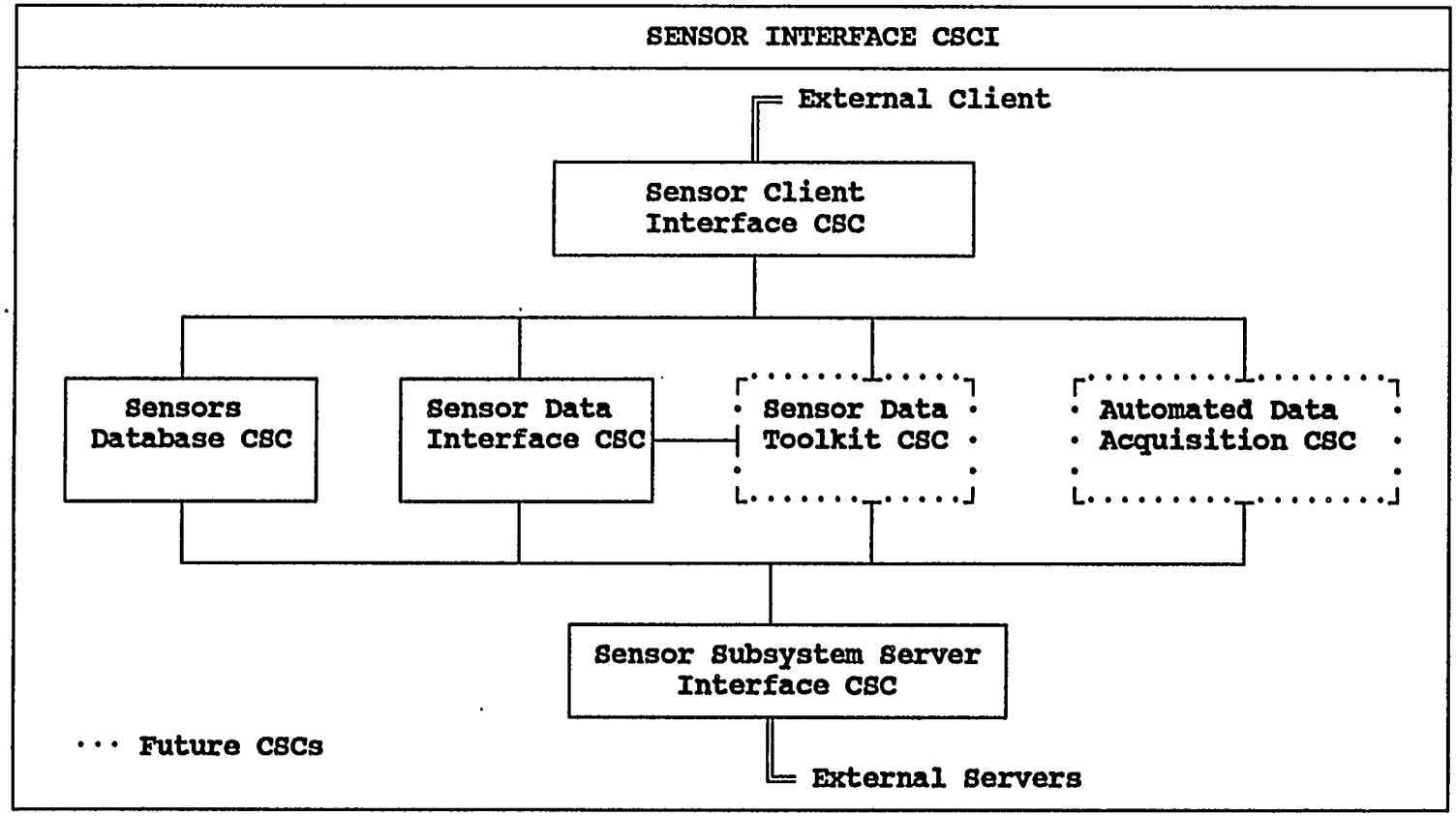


Figure 4-7 Sensor Interface CSCI Block Diagram

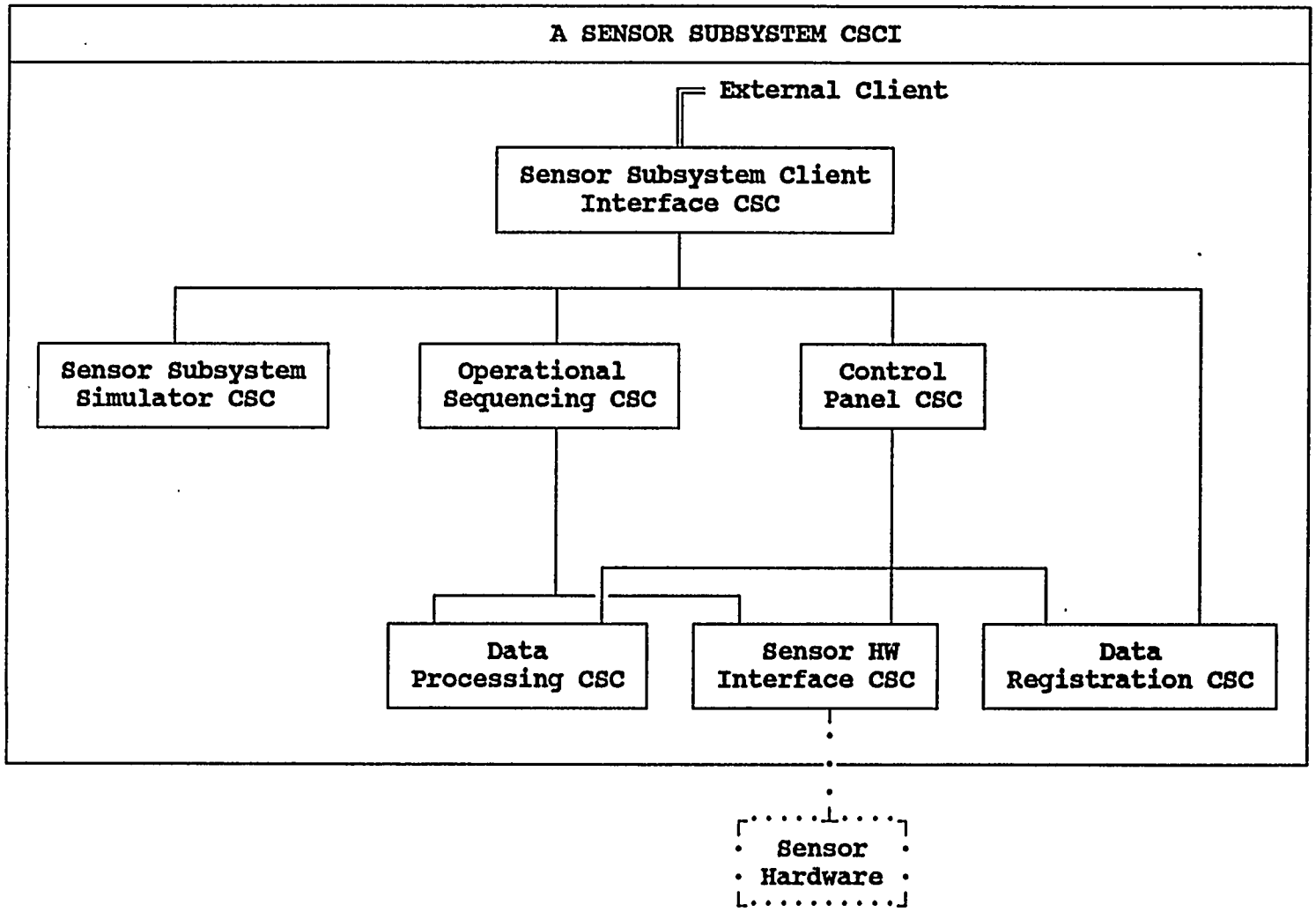


Figure 4-8: Generalized Sensor Subsystem CSCI Block Diagram

is not a required architecture for a sensor subsystem. The number of CSCs implemented and their functionality will vary for each sensor subsystem. The only requirement imposed by ICERVS is that the interface to the sensor subsystem comply with the Common Interface for Sensors specification.

4.2.4.1. Structured Light Sensor CSCI

As part of CRADAs with Sandia and Oak Ridge, MTI is developing a 4" diameter, single station version of a structured light sensor, including the controller for the sensor. This CSCI integrates the controller, the controller software and the ORNL developed mapping sensor control software. This CSCI will reside on the Sun Classic and will have been developed under the CRADA II effort.

4.2.4.2. Laser Range Finder CSCI

Only the Sensor Subsystem Simulator CSC will be implemented for this CSCI. The CSCI will simulate the PNL developed laser range finder sensor on the Sun Classic. MTI will design a remotely interfaced CSCI in accordance with the ICERVS Common Interface For Sensors standard. If available, actual measurement data from PNL will be incorporated into the simulation; otherwise, a mathematical model will be used to generate appropriate data. If actual measurement data are used, this CSCI will implement the necessary data transformation and registration algorithms to make the data compatible with the volumetric database. At a future point, these algorithms will be considered for incorporation into the Sensor Data Toolkit CSC of the Sensor Interface CSCI.

4.2.4.3. Sensor MiniLab CSCI

Only the Sensor Subsystem Simulator CSC will be implemented for this CSCI. The CSCI will simulate MiniLab on the Sun Classic. MTI will design a remotely interfaced CSCI in accordance with the ICERVS Common Interface For Sensors standard. It is expected that test data from INEL will be available in November 1993, and will be incorporated into the simulation; otherwise, a mathematical model will be used to generate material property data, such as temperature or electrical conductivity. If actual measurement data are used, this CSCI will implement the necessary data transformation and registration algorithms to make the data compatible with the volumetric database. At a future point, these algorithms will be considered for incorporation into the Sensor Data Toolkit CSC of the Sensor Interface CSCI.

4.3. Allocation of System Requirements

Table 4-1 assigns the system requirements to the various CSCIs and the structured light HWCI.

Table 4-1 Allocation Of System Requirements To ICERVS Phase II CSCIs

CSCI	Full Implementation	Partial Implementation	Phase II Design Influence
Phase II Demonstration CSCI	R3.12, R6.01, R6.04, R6.05, R6.07, R6.08, R6.09, R8.01		R8.02, R8.03
Volumetric Data CSCI	R1.01, R1.03, R1.04, R1.05, R1.06, R1.07, R1.08, R1.11, R2.01, R2.02, R2.05, R2.06, R2.07, R2.08, R3.01, R3.02, R3.03, R3.04, R3.05, R3.06, R3.07, R3.08, R3.09, R3.10, R3.11, R3.13, R5.01, R5.02, R5.03, R5.04, R5.05, R5.06, R5.07, R5.08, R5.11, R6.02, R6.03, R7.06 R7.07	R1.02, R2.03, R1.10, R3.14	R1.09, R2.04, R5.09, R6.06
Sensor Interface CSCI	R7.01, R7.02, R7.03, R7.04, R7.05, R9.07	R5.10	
Structured Light Sensor CSCI	R9.01, R9.04, R9.05, R9.06	R9.02, R9.03, R5.10, R7.08	R4.02
Laser Range Finder Simulator CSCI		R5.10, R7.08	
MiniLab Simulator CSCI		R5.10, R7.08	
Structured Light Sensor HWCI		R4.01, R9.10, R9.11	R4.03, R4.04, R10.01, R10.02, R10.03, R10.04

5. CLIENT-SERVER COMMUNICATIONS DESIGN

The ICERVS Phase II system embraces a client-server architecture. Processes (UNIX executable programs) called servers provide a particular service (set of functions) to other processes called clients that use that service. Client and server processes need not execute on the same computer platform. This section describes the design of the client-server communications interface that ICERVS will use.

5.1. General Design

The general design of ICERVS is that of one or more client processes (Demonstration Application, VDS User Interface, etc.) that require connection to one or more server processes (VDS, Sensors, etc.). The clients may reside on the same system as the server or different systems. (For Phase II, the clients and the VDS Server will reside on the SGI, while the sensor subsystems will reside on the SUN.) UNIX sockets, using Internet Protocol (IP), and Sun RPC will be used for communications. In addition, in order to attain GISC compatibility, the Sandia developed GENISAS and ISOE software will be incorporated into ICERVS.

While the Phase II implementation of the client-server interface will allow only a single client for each server, the client-server interface design will allow several future expansions, including:

1. Multi-client servers
2. A process can act as both a server and a client

5.2. Command Message Protocol

The command message protocol is concerned with the transmission and receipt of complete command messages. A command message contains three parts (fields): command name, number of arguments, and a list of arguments, as follows:

<cmdName> <narg> <arg1> <arg2> <arg3>, for example:

ConnectToServer 2 My Name My Data Path

All command fields are ASCII text strings to promote transparency between computer platforms. Note that the command protocol does not impose any restrictions on the argument values; single word and/or multi-word arguments are permitted. Command messages are best viewed as a data structure, such as:

```
struct COMMAND {  STRING  commandName;
                  STRING  numberOfArguments,
                  STRING  argumentList[MAX_ARGUMENTS]
               };
```

Every command message generates a reply. Reply messages are formatted exactly like command messages. The first argument is always a status word that indicates the success/failure of the command. For ICERVS the **replyName** field is always the keyword "REPLY".

<replyName> <narg> status <arg2> <arg3>, for example:

REPLY 2 0 USER ID #1

5.3. Communications Protocol

An underlying communications protocol is used to send / receive command messages. The command message protocol will encode/decode commands to/from a command packet (e.g., the structure above). The message protocol is only responsible for the actual transmission / receipt of messages. The command message protocol expects the communications protocol to reliably exchange command messages between the client and the server.

The communications protocol is concerned only with the exchange of messages. No attention is paid to the contents of messages. The only requirement is that all message elements be ASCII strings. For ICERVS, a UNIX socket-based approach is used. Sockets are straight forward to use and supported on all UNIX systems.

5.4. Implementation Approach

The initial implementation of the communications protocol will use software developed for the structured light mapping sensor CRADA. This software only need the encoding/decoding to be fully compliant with ICERVS requirements.

Later stages of software implementation will substitute the ISOE software at the communications protocol level. This change should be completely transparent to the command message protocol. In addition, the expanded capabilities of ISOE will allow ICERVS to add event, alarm, and data transfer protocols (needed by sensors) to the client server interface.

Finally, in the later stages of Phase II or in Phase III, the GENISAS package will be integrated into the client-server communications. This step will give ICERVS compatibility with DOE's GISC architecture.

5.5. Class Descriptions

CServerCommand: This class encapsulates the command data structure described in paragraph 5.2. It hides the details of argument buffer allocation / deallocation and provides the necessary class methods for assigning and retrieving command argument values.

CServer: This class represents a base class for defining server classes. It contains a command table (CServerCommand objects and function pointers) and is responsible for implementing the server side of the command message and communications protocols. Class methods for starting the server process, accepting clients, receiving command messages, dispatching commands for execution, and transmitting reply messages are included. Derived classes define the command table and provide the execution functions for each command. It is expected that ICERVS servers will derive from CServer.

CClient: This class represents a base class for defining client classes. It is responsible for implementing the client side of the command message and communications protocols. Class methods for starting the initializing the client, connecting to the server, transmitting commands, and receiving reply messages are included. Derived classes add a set of methods that provide access to the server's command set. It is expected that ICERVS clients will derive from CServer.

CUser: Future requirements envision servers with multiple simultaneous clients. To manage a set of clients and client context must be maintained. The CUser class represents a base class for that context. It contains all the client specific data and actually implements most server commands on behalf of its associated client. A new CUser object is created by the CServer object whenever a new client connects to the server.

5.6. Major Function Descriptions

Connect A Client: Connection of a client involves two parts: 1) making a socket connection 2) creating a new CUser object. The first step is handled by the standard UNIX sockets library functions (listen() and accept()). The second step requires the receipt of a ConnectToServer command. The CServer object receives the command (in CServer::DispatchCommands() method), creates a new CUser object, and passes the command to the CUser object for execution. The CUser object initializes its data tables for the new client and issues a reply message.

Disconnect A Client: This function is executed upon receipt of a DisconnectFromServer command. The CServer object deletes the appropriate CUser object, sends a reply message, and closes the client's socket. The CUser class destructor should perform all required shutdown / cleanup operations.

Receive and Execute A Command: Once a client has been accepted (socket library functions listen() and accept()), the CServer object issues socket library calls (recv()) to await receipt of a socket message. The message is parsed into a set of strings and a CServerCommand object is created. The client's identity is determined and the CServerCommand object is passed to the CUser object for that client (ExecuteServerCommand() method). The CUser method formats a reply message and returns to the CServer::DispatchCommands() method. The CServer object transmits the reply message to the client. The CServer object then issues another call to await the next message.

THIS PAGE INTENTIONALLY LEFT BLANK

6. ICERVS PHASE II DEMONSTRATION CSCI DESIGN

The ICERVS Phase II Demonstration CSCI is the topmost application level process and contains a high-level operator interface to demonstrate the functions and features of the ICERVS Phase II system.. This CSCI is representative of a user application and may be used as a template for creating a user application. The CSCI is composed of one CSC (Toplevel User Interface CSC) and the Application Data Structure. This section details the design of the software for this CSCI.

6.1. General Design Approach

The general design of this CSCI is that of a client application that requires connection with one or more UNIX-based server processes. The servers may reside on the same system as the client application or different systems. (During Phase II, clients and servers will execute on the same computer system). The connection and communications must be as universal and transparent as is possible. UNIX sockets, using Internet Protocol (IP), and Sun RPC have been chosen. In addition, the Sandia Intelligent System Operating Environment (ISOE), which is based on sockets, will be used to provide the low-level client-server support. Section 5.0 of this document details the client-server communications design and the classes defined in that section will be used to facilitate the implementation of the main program for this CSCI.

The largest volume of code for this CSCI will be for the windows, menus, buttons, and other GUI elements of the Toplevel User Interface CSC. However, the Uniras UIMX GUI builder tool will do most of the work and actually generate most of the code; which is good, since this CSC and its constituent Human Interface Component (HIC) classes tend to be rather volatile. Most of these HIC classes will interact directly with the Motif window system and possess callback methods to support the GUI. The menu and button callback routines for the HIC classes are fairly simple and are usually implemented by making a few functions calls to one or more of the Problem Domain Component (PDC) classes.

The analysis and design of this CSCI (as well as for all the other CSCIs) begins with the identification of the major Problem Domain Component (PDC) classes. An analysis of the requirements for this CSCI readily leads to the identification of the following major PDC objects (i.e. software classes). The relationships among these classes are illustrated in the OOA diagram of Figure 6-1.

Waste Site	Sensor	Sensor/Dataset Connection
Waste Site Log	Camera	Camera/View Connection
	Dataset	View
	Property Type	

As can be seen from the OOA diagram, the Demonstration CSCI is primarily concerned with Waste Sites and their associated Log, Datasets, and Sensors. A Dataset is associated with Sensors and Views. A Camera is a specialization of a Sensor. Sensors are associated with a Material Property Type. A View is associated with a Camera. A Sensor can send data to its associated Dataset. A Camera can send its position and orientation to a View so that the View can synchronize its display with the Camera.

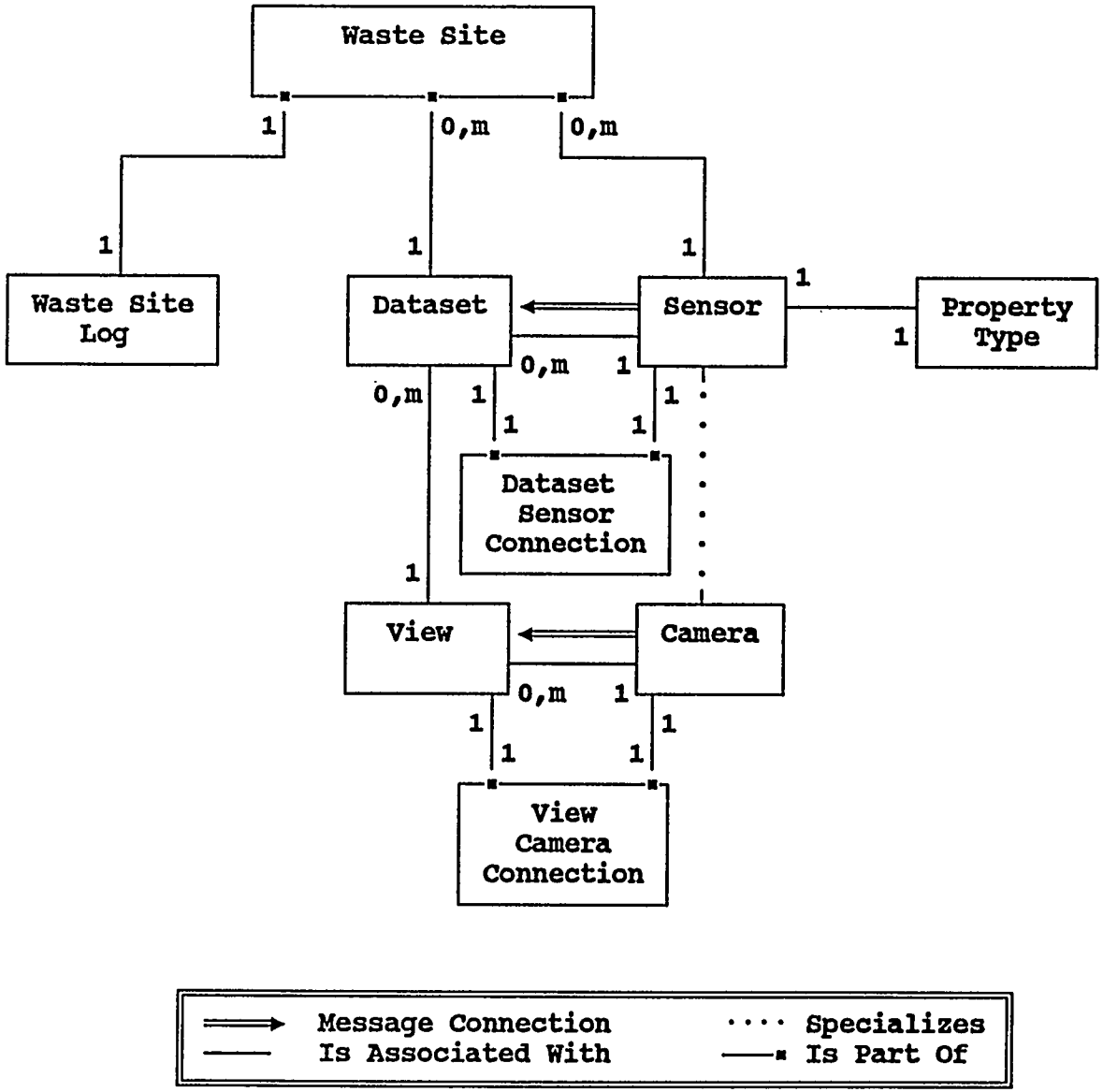


Figure 6-1 -- Demonstration CSCI OOA Diagram

6.2. Requirements Allocation

The requirements for each of the CSCs are summarized in Table 6-1. (Since only one CSC exists, all requirements are assigned to it.) For a more detailed description of each requirement, refer to Appendix D of the *ICERVS Phase II System Design Report.* Requirements that apply (full or partial implementation) to ICERVS Phase II, including those from Phase I, are in bold type. Other requirements that are not part of Phase II but strongly influence the software design are *italicized.*

Table 6-1 Requirements For Toplevel User Interface CSC

Requirement Number	Description
R3.12	**Set view parameters to track sensor station viewpoint**
R6.01	**Operator edit system parameters**
R6.04	**Maintain operator log and operator note book for observations or other notes**
R6.05	**Support multiple systems of units**
R6.07	**Establish system data structure**
R6.08	**Support multiple world models / data sets per waste site**
R6.09	**Support multiple waste sites per system**
R8.01	**Graphic tools: spaceball and/or mouse, pull down menus, dialog boxes**
R8.02	*Provide operator help facility*
R8.03	*Provide hard copy output*

6.3. Operator Interface Description

The ICERVS Phase II operator interface begins with the main window. The ICERVS main window is the top level window and remains on the screen until the program is exited. All other windows created will be displayed on top of this window. The Phase II system functions are implemented via pull-down menus from the main window. Figure 6-2a illustrates the main window and its menu functions.

6.3.1 Menu Function Descriptions

Each menu and its menu functions are described briefly below:

SYSTEM MENU

Login *(Login as ICERVS operator)*: Displays an input dialog that prompts for operator name and password. Upon validation of the operator name and password, makes that person the current ICERVS operator.

About: (*Display ABOUT ICERVS information*): Displays an ABOUT message for the ICERVS system in a window.

Quit *(Exit ICERVS system)*: Closes all window, releases all resources and exits the ICERVS Phase II Demonstration CSCI.

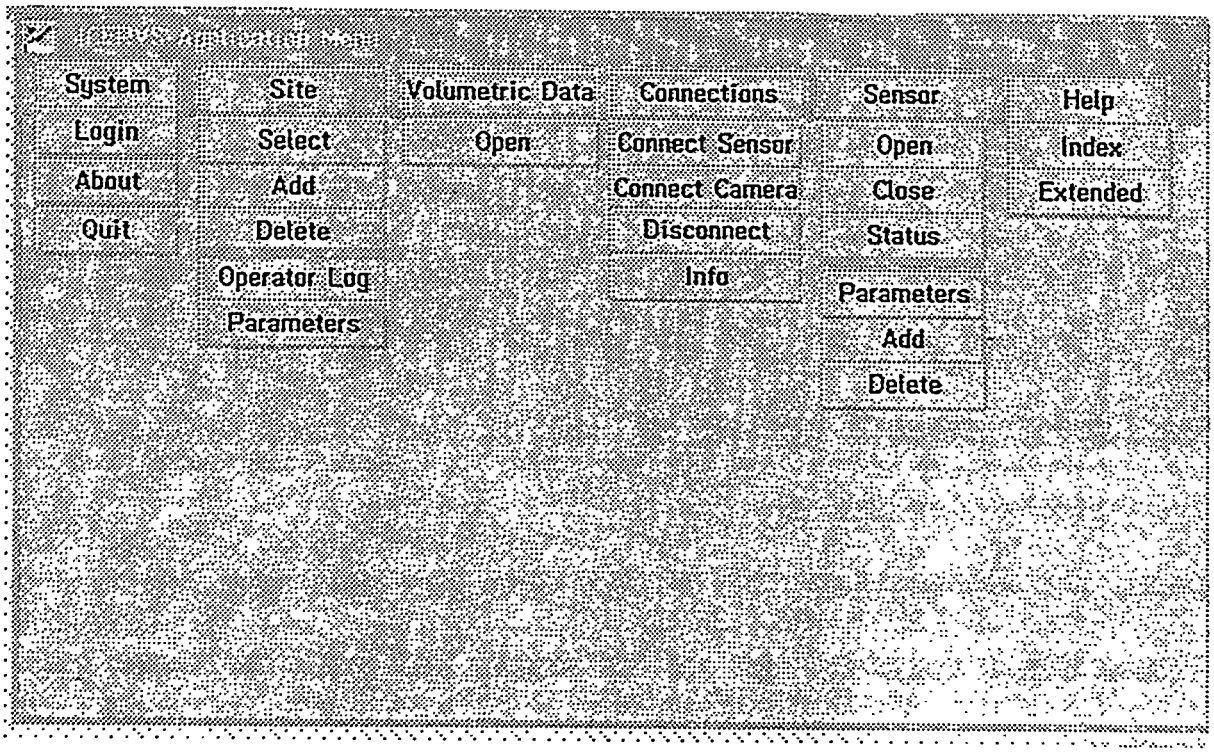


Figure 6-2a ICERVS Phase II Main Window Screen

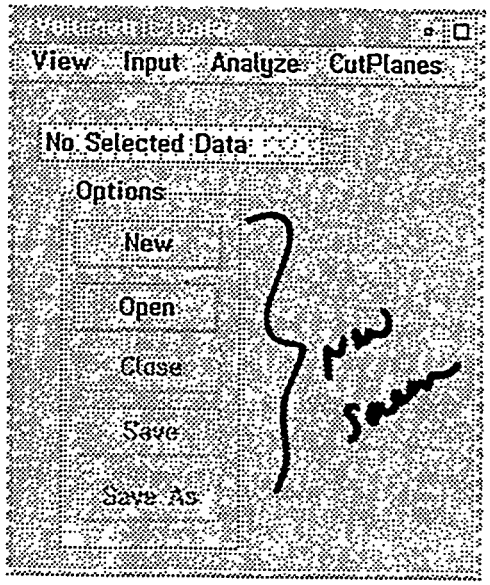


Figure 6-2b Volumetric Data Window

SITE MENU

Select *(Select a current site)*: Allows operator to select an existing site to become the default current site. Many of the menu functions on the Main Menu require specification of a site. If a current site is defined, these functions do not request a user specified site.

Add *(Create a new site)*: Creates a new site directory with default files.

Delete *(Delete an existing site)*: Removes an existing site directory and its files.

Append Log *(Add a new entry to site log file)*: Displays a blank edit window in which the operator may enter any information desired. The information is then formatted into a log entry and appended to the site log file.

Print Log *(Print the site log file)*: Allows viewing and printing of the a site log file.

Edit Parameters *(Edit the site parameters file)*: Allows modification of a site parameters file via a scrollable, multi-line edit window.

Print Parameters *(Print the site parameters file)*: Allows viewing and printing of the a site parameters file.

VOLUMETRIC DATA MENU

Open *(Open Volumetric Data Window)*: Establishes a client-server link with the ICERVS Volumetric Data Server. If opened in interactive mode (the only mode supported for Phase II), the server creates a Volumetric Data Window on the client's display (the SGI). Once a data set has been selected, various viewing, input, analysis, and cut plane functions can be performed on the data set. Figure 6-2b illustrates the Volumetric Data Window.

CONNECTIONS MENU

Connect Sensor *(Connect a sensor to a dataset)*: Displays a control panel that allows connecting an active sensor to a selected volumetric dataset for a particular site. Once connected, data can be transferred from the sensor to the data. Figure 6-3 illustrates the sensor connection control panel.

Connect Camera *(Connect a camera to a view of dataset)*: Displays a control panel that allows connecting an active camera to a selected view of a volumetric dataset for a particular site. Once connected, the view can be commanded to track with the scene from the camera. Figure 6-4 illustrates the camera connection control panel.

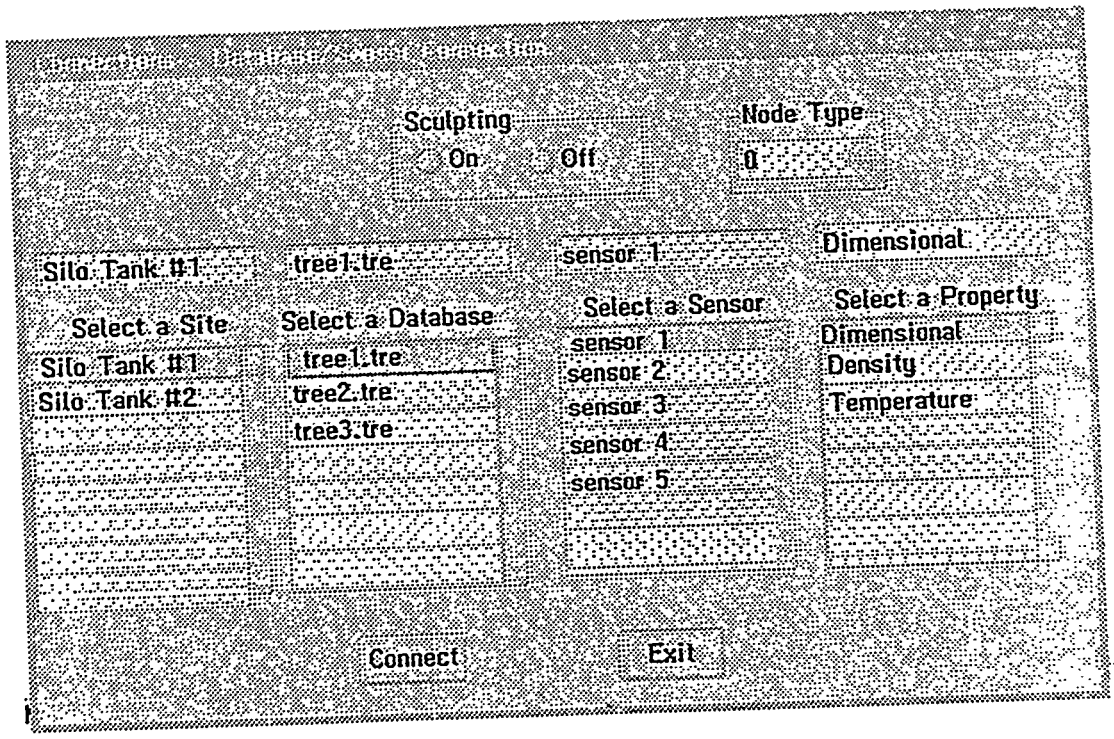


Figure 6-3 Sensor Connection Panel

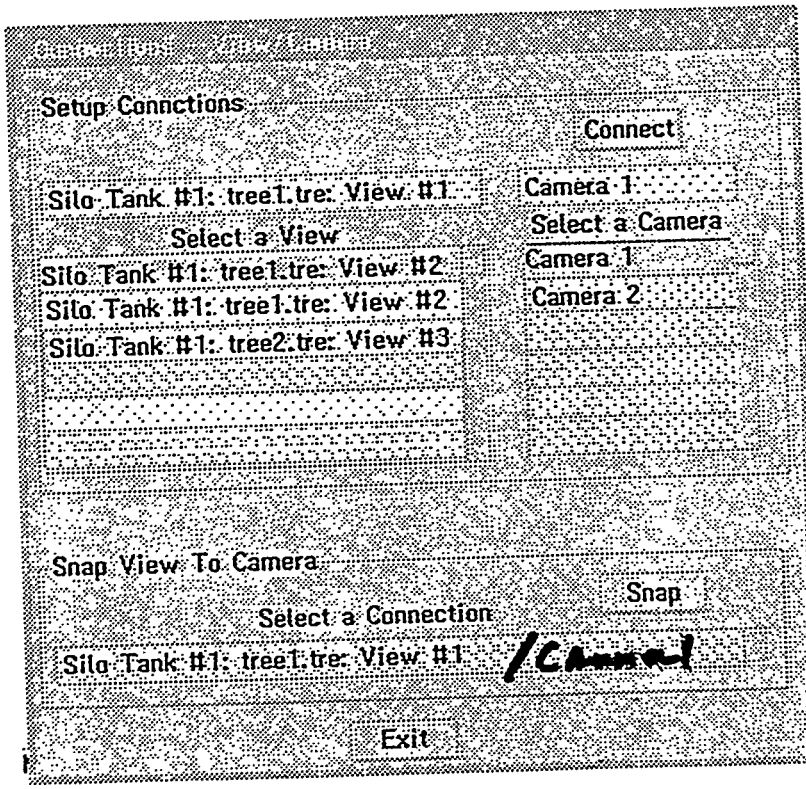


Figure 6-4 Camera Connection Panel

Disconnect *(Disconnects a sensor or camera from a dataset)*: Breaks a previously defined connection between a sensor/dataset or camera/view.

Information *(Display connection information)*: Displays a window that shows information about the currently defined connections. The data may be optionally printed.

SENSOR MENU

Open *(Open a sensor for data acquisition)*: Establishes a client-server link with the ICERVS Sensor Interface Server and the server for a selected sensor for a particular site. The sensor is declared active. If opened in interactive mode (the only mode supported for Phase II), the sensor server will open a control panel window on the client's display (the SGI). The sensor may then be configured and manipulated via its control panel. A sensor control panel may be closed without deactivating its associated sensor. Control panels for multiple sensors may be open on the client screen at the same time.

Close *(Close currently open sensor)*: Dissolves the client-server link with the selected sensor. The sensor control panel will be closed and the sensor will become inactive. Any data link connections must be dissolved first.

Add *(Add a new sensor to a site)*: Allows the operator to add a new sensor to a selected site. An editor window is automatically brought up to allowing modification of the site-specific sensor parameters file.

Delete *(Delete sensor from a site)*: Allows the operator to delete a sensor from a selected site.

Edit (Edit site specific sensor parameters file): Allows modification of a site specific sensor parameters file via a scrollable, multi-line edit window.

Print (Print site specific sensor parameters file): Allows printing of the a site specific sensor parameters file.

Status *(Display information about sensors)*: Displays a window that shows information about the currently defined sensors for a site. The data may be optionally printed.

HELP MENU

Index: *(Display HELP index)*: Displays an index of HELP information in a selection dialog window. When the operator selects the desired HELP index item, all information about that item will be displayed in a scrollable window. This function is not implemented in Phase II.

Extended: (*Display extended HELP for the system*): This function displays a scrollable multi-line window containing system wide HELP information. This function is not implemented in Phase II.

6.3.2. UIMX Interface Details

The Uniras GUI builder utility (UIM/X) will be used to design and layout the various user interface screens. This utility allows interactive creation, deletion, modification, placement, and movement of the various X widgets that comprise a particular user interface screen. When the layout stage is completed, UIMX can generate the C-language source code for the interface screen. UIMX can generate code using Uniras library calls or Xt calls (actually Motif, Xtoolkit and Xlib calls). For ICERVS Phase II, all user interface screen source code will be generated for Xt in order to promote portability among UNIX systems. The ICERVS Application CSCI user interface consists of the components illustrated in Figure 6-5. Some of the components are general purpose (SelectionBox, MessageBox, Browser, Editor, RequestBox, and FileListBox) and others are specific to this CSCI (CameraConnectionPanel. SensorConnectionPanel, CameraViewSnapPanel). The HelpPanel will be only partially implemented in Phase II.

Since UIMX generates C code and ICERVS uses C++ code, encapsulating C++ classes will be required in order to isolate the UIMX code. This encapsulation process is fairly straight forward. However, there are a few areas that merit special consideration. The following paragraphs discuss these areas.

6.3.2.1. Create and PopUp Functions

UIMX supports two types of public interfaces for the code that it generates:

Create Interface: The generated UIMX code only creates the user interface screen and all of its components. The user interface screen is not managed nor displayed. It is the callers responsibility to manage and display the user interface screen. Since UIMX returns a pointer to the topmost Widget in the user interface screen, the caller may make additional Xm, Xt, or Xlib calls to customize the screen. The caller can use the UIMX UxPopupInterface utility function to display the screen.

Popup Interface: The generated UIMX code creates the user interface screen and all of its components. The UIMX code then internally invokes UxPopupInterface to display the user interface screen. The caller has no opportunity to customize the screen.

The ICERVS Phase II software will utilize the **Create Interface.**

Furthermore, ICERVS requires that each create interface accept two arguments: **parent** and **object.** An example of a create interface function is: - **create_SomeUserScreen(Widget parent, void* object).** The UIMX code will optionally use the parent argument, but will retain copies of both arguments in its context structure for the user screen. The object argument points to the instance of the C++ class that encapsulates the UIMX code and will be used during event callbacks.

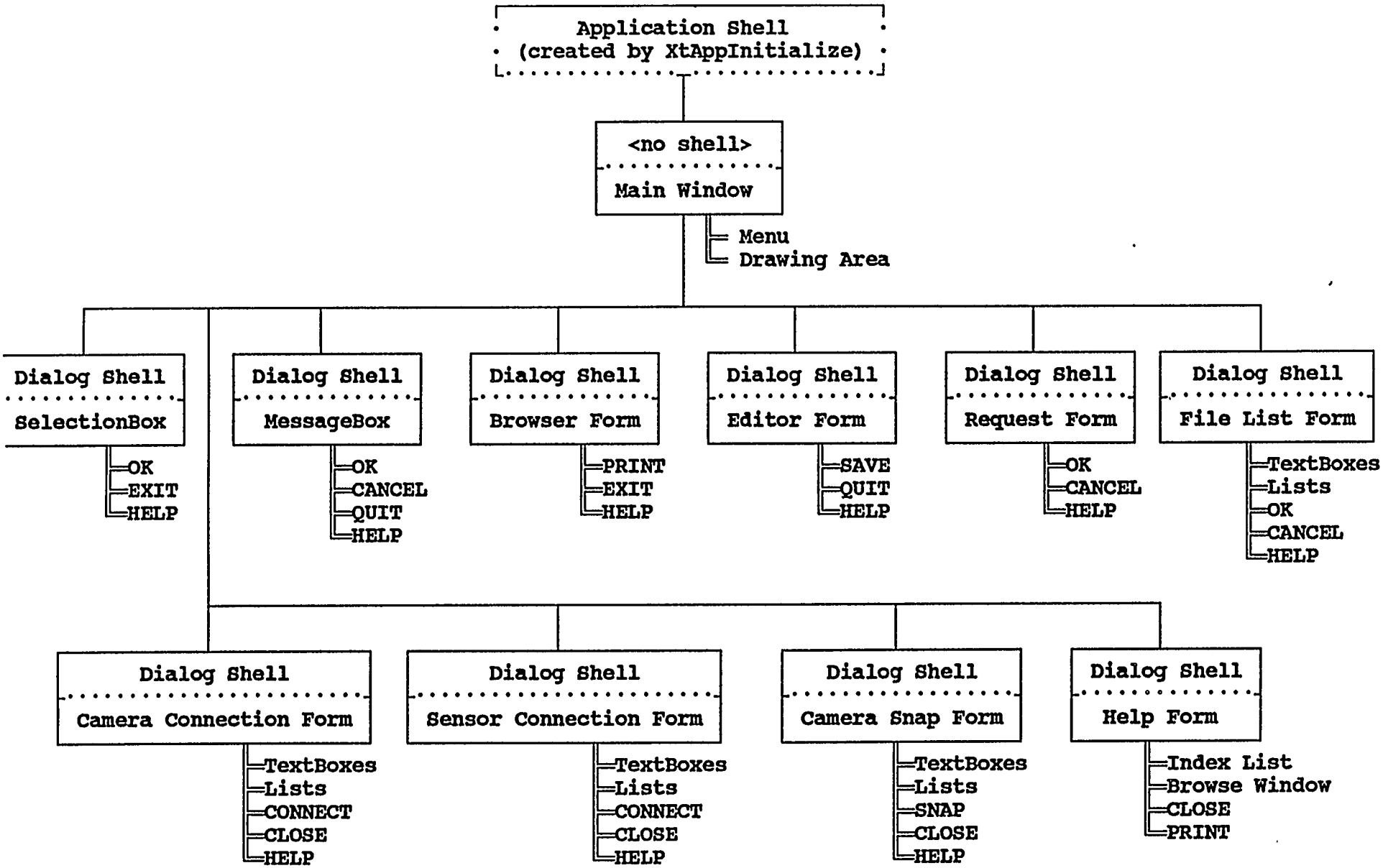


Figure 6-5: ICERVS Phase II Demonstration CSCI
User Interface Components

6.3.2.2. Callback Functions

The ICERVS user interface, like all GUI interfaces, is event driven. That means that all processing is the result of some action (button click, mouse movement, keyboard entry, etc.) by the user. The association between the user's action and the application software is implemented with callbacks. Each action for each object in the user interface has an associated body of software logic (a callback function) that the windowing system (X-Windows) activates in response to a user action. These callbacks are received by the UIMX generated code and usually need to be relayed to the C++ application code.

However, C-language functions (the UIMX code) cannot directly invoke C++ class methods (the ICERVS application code). The key to solving this problem lies in the encapsulating C++ class.

1. The encapsulating class will pass its pointer (the **this** pointer) to UIMX via the create interface function.

2. The encapsulating class will contain one or more C-callable, global helper functions that can be invoked by UIMX.

3. The UIMX code will pass the object pointer to the helper function during an event callback.

Since most of the screens in the ICERVS user interface use one or more of a small set of buttons, a standard set of helper functions has been defined:

```
extern void CB_ApplyCallbackHelper(void* object);
extern void CB_CancelCallbackHelper(void* object);
extern void CB_ExitCallbackHelper(void* object);
extern void CB_HelpCallbackHelper(void* object);
extern void CB_OkCallbackHelper(void* object);
extern void CB_PrintCallbackHelper(void* object);
extern void CB_QuitCallbackHelper(void* object);
extern void CB_SaveCallbackHelper(void* object);
extern void CB_GenericCallbackHelper(void* object, int action);
```

User interface screens that contain menus or return values require a specialized callback helper interface.

```
extern void CB_MainWindowCallbackHelper(void* object, char* callbackName);
extern void CB_SelectionCallbackHelper(void* object, char* selectedText);
extern void CB_RequestDialogCallbackHelper(void* object, char** userInputs);
```

Complex user interface screens (CameraConnectionPanel, SensorConnectionPanel) will require additional specialized callback helper routines.

6.3.2.3. Menu Button Disabling/Enabling

Screens with menus will need a mechanism for enabling and disabling menu buttons. When a menu function is selected, its menu button must be disabled until the operation is completed. Other menu buttons on the same pop down menu should remain enabled. (In some cases, other menu buttons may be disabled.) The following protocol will be implemented for ICERVS to address this issue:

1. The UIMX menu button callback routine will disable the menu button every time the callback is invoked. Consequently, when the C++ class method is invoked, the menu button will have already been disabled.

2. The UIMX code will include an externally callable function for enabling and disabling the menu buttons. This function will have a similar name to the UIMX create function.

3. An encapsulating C++ class method will call the menu button function whenever necessary to enable or disable a menu button.

6.3.2.4. An Interface Example

As an example to illustrate the concepts and protocols defined above, consider the C++ interface to a screen that has a menu and an OK button.

Create Interface:

```
create_SomeScreen(Widget parent, void* objectPointer)
{UIMX generated code
return(theWidget)}
```

UIMX Callback:

```
UIMX_CallbackFunctionForOkButton(Widget, ClientData, CallbackData)
{UIMX entry code to set context
CB_OkCallbackHelper(objectPointer);
UIMX exit code }
```

UIMX Callback:

```
UIMX_MenuButtonCallbackFunction(Widget, ClientData, CallbackData)
{UIMX entry code to set context
XtSetSensitive(MenuButtonWidget , False);
CB_MainWindowCallbackHelper(objectPointer, "Menu Button #1");
UIMX exit code}
```

Menu Button Function:

```
menubutton_SomeScreen(Widget screen, char* buttonName, boolean state)
{UIMX type logic to get context for Widget screen
if (strcmp(buttonName, "Menu Button #2")
   XtSetSensitive(MenuButton2Widget , state);
else if (strcmp(buttonName, "ANOTHER BUTTON")
   XtSetSensitive(OtherButtonWidget, state);
UIMX type logic to restore previous context}
```

6.3.2.5. Higher Level Callbacks

Paragraph 6.3.2.2 described the callback interface between the UIMX C-language code and the ICERVS C++ code of the encapsulating class. That interface was based upon the existence of C-callable, global helper functions in the encapsulating C++ class. A helper function's main purpose is to provide a bridge to the appropriate encapsulating class method to handle the callback. An example implementation of a callback helper function for a class called ABC might look like the following:

```
CB_PrintCallbackHelper(void *object)
{
  ABC* abc = (ABC *) object;          //Cast to a pointer to an object of class ABC
  abc->StdPrintCallback();            //Now call the real handler method
}
```

In order to facilitate the callback interface, three classes have been defined:

CStandardCallbackFacility: A class that encapsulates the standard callback interface with the UIMX code. This class implements the standard callback helper functions identified in paragraph 6.3.2.2. This class also provides default implementations (as virtual methods) for the class methods that correspond with each helper function. These class methods are:

StdApplyCallback	StdHelpCallback	StdQuitCallback
StdCancelCallback	StdOkCallback	StdSaveCallback
StdExitCallback	StdPrintCallback	StdGenericCallback

Since the standard callback methods are implemented as virtual methods, a derived class simply implements an overriding method in order to capture the callback for itself. Not all of the standard callback methods need to be overridden; those not overridden will be handled by the CStandardCallbackFacility base class.

All classes that wish to interface with the UIMX code via the standard callbacks must derive from the CStandardCallbackFacility class.

CCallbackBase: A base class for all classes wishing to request user callbacks from a CUserCallbackFacility class.

CUserCallbackFacility: A class that encapsulates a callback interface between the CStandardardCallbackFacility class and higher level user classes that are derived from CCallbackBase. This class allows the higher level class to define user level callback functions that can be invoked from the CStandardCallbackFacility standard callback methods. It is completely at the discretion of the CStandardCallbackFacility derived class as to whether it wishes to support user level callbacks.

Figure 6-6 illustrates the propagation of a PRINT button press event through the various layers of software. The action begins within the X-Windows system when the PRINT button press is detected. A callback is made to the UIMX callback function (PrintButtonCallback) for the button. The UIMX function calls a global helper function (CB_PrintCallbackHelper) of class ABC, which is derived from class CUserCallbackFacility. The helper function relays the callback to the ABC::StdPrintCallback method. ABC::StdPrintCallback performs whatever actions are appropriate and invokes a previously defined user level callback made by an instance of class USER, which is derived from CCallbackBase.

6.4. Application Data Structure Details

The data manipulated by the ICERVS Volumetric Data Server and ICERVS Sensor Interface Server belong to the end user and are stored in some user data storage area. When an end user activates the ICERVS package, the location of the user's data must be specified. Furthermore, in order for the ICERVS servers to access and manipulate the data, the organization and contents of the Application Data Structure must be standardized. The general organization and contents for the Application Data Structure is shown in Table 6-2:

6.4.1. Definition Of Files

The files stored in the Application Data Structure are described below. Appendix ?? contains examples of the contents of each file.

System Level Files:

Help Text File (help.txt): This is an ASCII file containing the ICERVS system HELP file. Its maintained external to the ICERVS by any general purpose text editor. There is only one help file per ICERVS system.

Master Dataset List File Template (dsetdir.def): A version of A Dataset List File that contains no dataset entries. It is used to initially create the a Dataset List File for a new site.

Master Object Templates File (object.mlb): This file contains the standard set of ICERVS geometric object templates. A copy of this file is placed in each new site directory and this copy may be altered as required.

Master Property List File (property.lst): This file contains a list of all known material property types. The file contains generic information (such as type, description, units label, etc.) that ICERVS uses to manage properties. The file is initially created by the ICERVS system manager and is maintained external to the ICERVS system with any general purpose text editor.

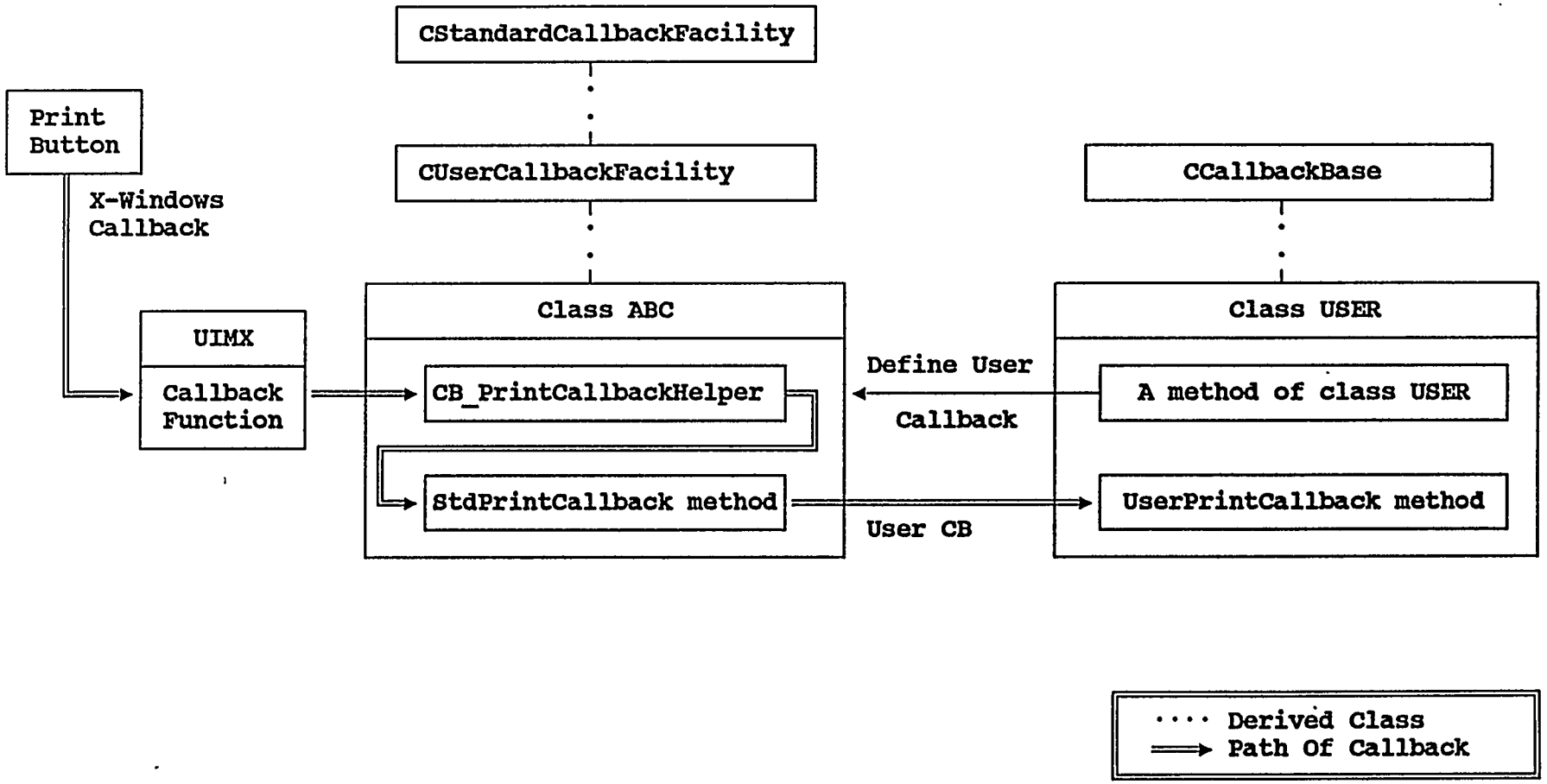


Figure 6-6: Flow Of Events During A Callback

Table 6-2 ICERVS Application Data Structure

/USR/SOMEWHERE/...	An arbitrary point in users disk hierarchy
/ICERVS	Root of ICERVS Data Structure
/etc	Contains templates and general files
dsetdir.def	Master Datset LIst File Template
help.txt	Help Text File
object.mlb	Master Object Templates File
property.lst	Master Material Property List File
sensor.lst	Master Sensor List File
sensor.def	Master Sensor List File Template
sitedir.lst	Master Site List File
sitelst.def	Master Site List File Template
siteprm.def	Master Site Parameters File Template
view.def	Default View Parameters (system-wide)
/SITE1	Contains files for first site
datset.lst	Datset List File
object.lib	Object Templates File
site.prm	Site Parameter File
sensor.lst	Sensor Parameter File
site.log	Site Log File
view.prm	Default View Parameters (site specific)
/DATASET1	Contains volumetric/geometric data files
xxxxx.tre	Volumetric Data Files
yyyyy.prp	Property Data Files
zzzzz.raw	Raw Data Files
object.dic	Geometric Object File
view.prm	Default View Parameters (dataset specific)
aaaaaa.vew	Saved Viewset File
/DATASET2	Contains another set of data files
.	
.	
/SITE2	Contains files for second site
.	
.	

Master Sensor List File (sensor.lst): This file contains a list of all defined sensors types in the ICERVS system. The file contains generic information (such as type, host machine, service name, etc.) that ICERVS uses to manage sensors. The file is initially created by the ICERVS system manager and is maintained external to the ICERVS system with any general purpose text editor.

Master Sensor List File Template (sensor.def): A version of the Master Sensor List File that contains no sensor entries. It is used to initially create the Master Sensor List File. A copy of this file is placed in each new site directory and this copy may be altered as required.

Master Site List File (sitedir.lst): This file contains a list of all defined sites in the ICERVS system and identifies the last selected site. The file is initially created by the ICERVS system manager. The file is automatically updated whenever sites are created or deleted.

Master Site List File Template (sitelst.def): A version of the Master Site List File that contains no site entries. It is used to initially create the Master Site List File.

Master Site Parameter File Template (siteprm.def): A default version of a site parameter file. This file is copied to the directory for a newly created site. The file is initially created by the ICERVS system manager and is maintained external to the ICERVS system with any general purpose text editor.

Default View Parameters File (view.def): This file contains a system-wide set of default parameters for view windows. A copy of this file is placed in each new site directory and this copy may be altered as required.

Site Specific Files

Dataset List File (dataset.lst): This file contains a list of all defined datsets for the site. The file is initially copied from the Master Dataset List File Template (dsetdir.def) and is subsequently maintained by the ICERVS system as datasets are added and deleted.

Default View Parameters File (view.prm): This file contains default values for view parameters. A copy of this file is placed in each new dataset subdirectory and that copy may be altered as required. ICERVS supports an editor that allows the file contents to modified as required.

Object Templates File (object.lib): Initially, this file is a copy of the Master Object Templates File (object.mlb). The file may then be tailored by adding, deleting, or modifying the objects contained within.

Sensor Parameter File (sensor.lst): This file contains all the site specific sensor parameters. It is initially created by copying the Sensor Parameter File Template. ICERVS Phase II supports an editor that allows the contents of the file to be updated.

Site Log File (site.log): This file is an ASCII text file that contains operator log entries for the particular site. No restrictions are placed on the content nor the format of entries. The ICERVS system will attach a date/time stamp and operator identification to each log file entry.

Site Parameter File (site.prm): This file contains all the site specific parameters. It is initially created when ICERVS creates a new site. ICERVS Phase II supports an editor that allows the contents of the file to be modified.

Geometric Objects File (object.dic): This file contains the user defined 3D geometric objects for a specific dataset for a specific site. ICERVS reads the file, displays the geometric objects, and permits creating, deletion, editing, and printing of objects.

Volumetric Data Files (xxxxxx.tre): This file(s) contains the volumetric data in octree form.

Property Data Files (yyyyyy.prp): This file contains the property data associated with an octree.

Raw Data Files (zzzzzz.raw): This file (or files) contain the raw data received from a sensor. It is stored in a sensor specific format and can used to recreate the volumetric dataset.

Saved Viewset File (aaaaaa.vew): This file contains the parameters that define a set of views including window location, view type, translation, scaling, rotation, colors, etc.

6.5. Toplevel User Interface CSC Detailed Design

The Top level User Interface CSC provides the user with access to the ICERVS Phase II main level functions. It incorporates some evolutionary upgrades (such as menu revisions) to the existing Phase I top level graphical user interface. It adds a new capability for generating and managing sensor windows and connections between sensors/cameras and datasets/views.

6.5.1. Class Descriptions

The Toplevel User Interface CSC software is objected oriented, implemented in C++ and consists of approximately 50 classes. This section identifies the CSC classes and discusses their general characteristics (attributes, behavior and relationships). Section 6.5.2 will discuss the major functions assigned to the CSC and describe how the software classes implement the functions.

In keeping with good object-oriented analysis practices, the classes are grouped by the categories Problem Domain Component (PDC) and Human Interface Component (HIC).

6.5.1.1. Problem Domain Classes

The OOA diagram of Figure 6-1 has been expanded to produce the OOD diagrams in Figures 6-7, 6-8, and 6-9. Figure 6-7 has added two new classes (CNamedItem and CConnectableItem) to serve as generalizations of the OOA model basic PDC classes. (The RWCollectable class is from the Rogue Wave Tools^{++} library.) Figure 6-8 introduces a set of classes that form collections of the basic PDC classes. (The RWOrdered class is from the Rogue Wave Tools^{++} library.) Figure 6-9 combines all the PDC classes and illustrates the Is A Part Of relationships among objects of the various classes.

The PDC classes for the Toplevel User Interface are identified and briefly described below.

CConnectableCamera: A class that represents a camera. The class is derived from CConnectableSensor.

CConnectableDataset: A class that represents a dataset that can be associated with a sensor. The class is derived from CConnectableItem.

CConnectableItem: An abstract base class that provides all the common characteristics and behavior for connectable items. This class is derived from the class CNamedItem. Since CNamedItem is derived from RWCollectable, all CConnectableItem derived classes are also collectable and may use the RWCollection classes.

CConnectableSensor: A class that represents a sensor. The class is derived from CConnectableItem.

CConnectableView: A class that represents a view that can be slaved to a camera. The class is derived from CConnectableItem.

CDataLinkConnection: A class that describes the connections among a set of CConnectableItem objects. In Phase II, instances of objects of this class are used to manage the connections between sensors/databases and cameras/views.

CGlobalProblemDomainComponentData: A class that holds all necessary application dependent global PDC data parameters. All data members of this class are static.

CNamedItem: A base class for things that have names and descriptions. This class also includes an identification number for its item. CNamedItem is derived from RWCollectable.

CMaterialProperty: A class that represents a material property. The class is derived from CNamedItem.

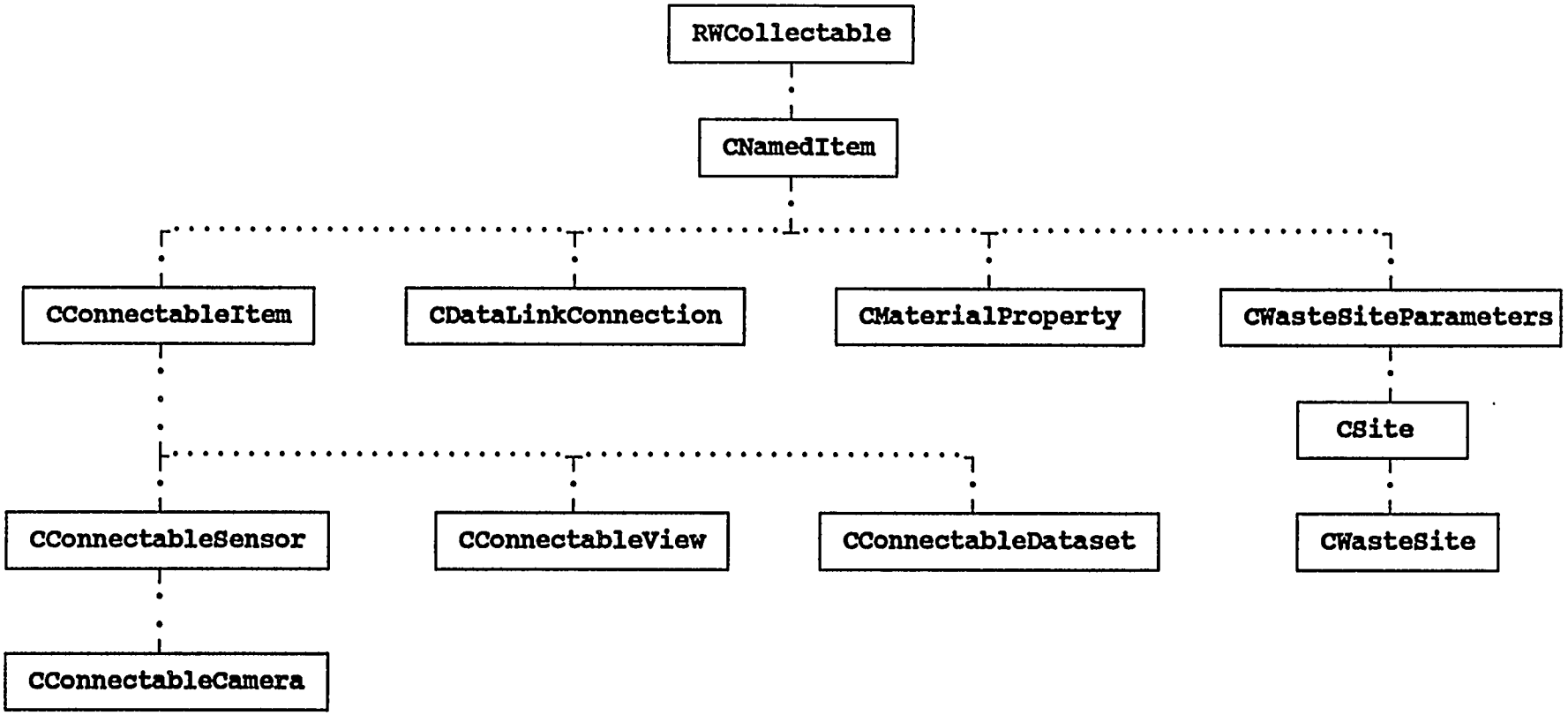


Figure 6-7: ICERVS Phase II Demonstration CSCI OOD Diagram #1

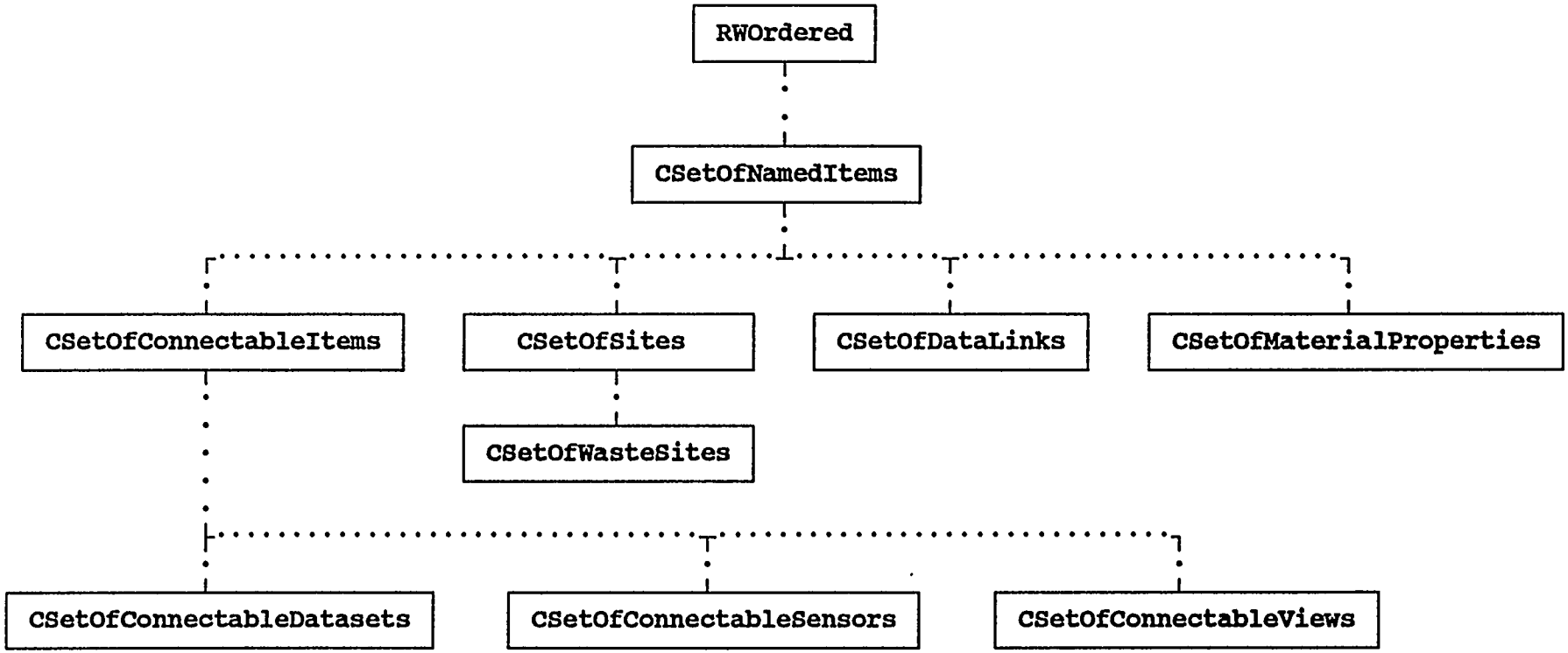


Figure 6-8: ICERVS Phase II Demonstration CSCI
OOD Diagram #2

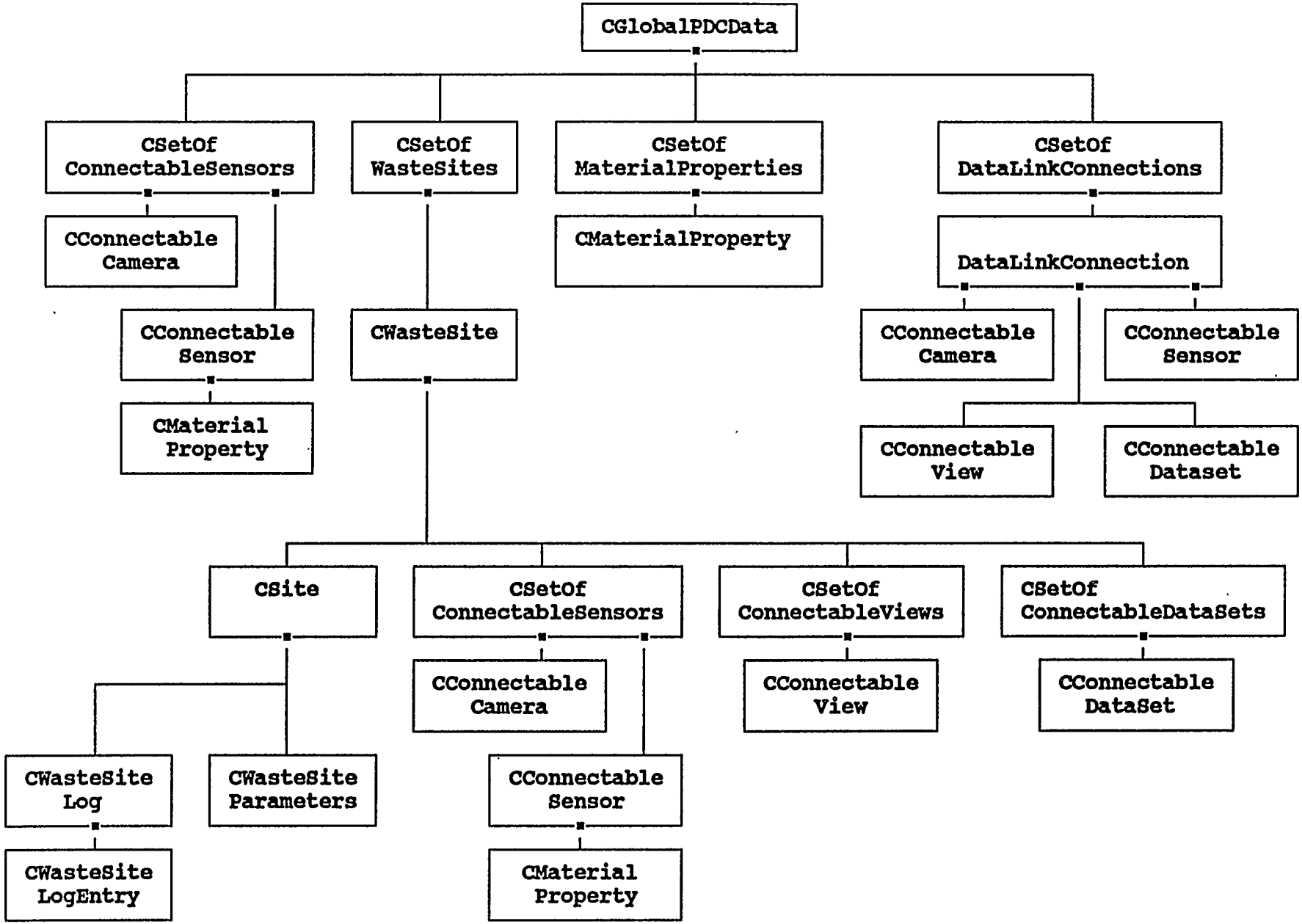


Figure 6-9: ICERVS Phase II Demonstration CSCI
OOD Diagram #3

CSetOfConnectableDatasets: A class that implements a collection of CConnectableDataset objects.

CSetOfConnectableItems: A class that implements a collection of CConnectableItem objects. This class is only used as a base class for specializing (derived) classes.

CSetOfConnectableSensors: A class that implements a collection of CConnectableSensor and CConnectableCamera objects. Three instances of this class are generally in use: a set of all sensors known to the ICERVS system, a set of defined sensors for a particular site, and a set of active sensors for a particular site..

CSetOfConnectableViews: A class that implements a collection of CConnectableView objects.

CSetOfDataLinkConnections: A class that implements a collection of CDataLinkConnection objects.

CSetOfNamedItems: A class that implements a collection of CNamedItem objects. This class is only used as a base class for specializing (derived) classes.

CSetOfMaterialProperties: A class that implements a collection of CMaterialProperty objects.

CSetOfSites: A class that implements a collection of CSite objects.

CSetOfWasteSites: A class that implements a collection of CWasteSite objects.

CSite: A class that encapsulates all the waste site (workspace or taskspace) specific data and control parameters. One instance of this class exists for each waste site known to the ICERVS system. The CSite class is derived from CSiteParameters class.

CWasteSite: A class derived from CSite that includes a set of sensors, a set of datasets, and a set of views that are associated with the site.

CWasteSiteLog: A class that encapsulates all aspects of the site log file.

CWasteSiteParameters: A class that encapsulates all aspects of the site parameter file and the data items contained in the file. Read and write capabilities are provided. This class is derived from the class CNamedItem. . Since CNamedItem is derived from RWCollectable, all CWasteSiteParameters derived classes are also collectable and may use the RWCollection classes.

6.5.1.2. Human Interface Classes

Once the PDC OOA and OOD processes had matured, OOA and OOD were applied to the Demonstration CSCI user interface. Figures 6-10 and 6-11 identify the major HIC classes and their inheritance hierarchy. Finally, Figures 6-12 and 6-13 combine the key aspects of the PDC and HIC classes and illustrate some of the associations between PDC and HIC objects.

The HIC classes for the Toplevel User Interface are identified and briefly described below. Much of the layout and code for these classes will be generated by the Uniras UIMX tool.

CAboutDialog: A specialized dialog box that displays descriptive text about ICERVS.

CBrowseWindow: A general purpose read only window class for viewing and/or printing an ASCII text file. This class will encapsulate the UIMX browse dialog form.

CCameraConnectionControlPanel: A specialized window for specifying connections between cameras and volumetric dataset views.. This class will encapsulate the UIMX camera connection dialog form.

CCameraSelectionListBox: A specialized selection list box that queries its CSetOfConnectableSensors object, generates a list of camera names, invokes a general selection list box, and returns the selected camera name.

CConnectionSelectionListBox: A specialized selection list box that queries its CSetOfDataLinkConnections object, generates a list of data link names, invokes a general selection list box, and returns the selected data link name.

CConnectionDeleteVerifyDialog: A class that displays a confirmation message to confirm the deletion of a connection from the system.

CConnectionSetStatusBrowser: A class that displays the set of connections and status information about each connection.

CDatasetSelectionListBox: A specialized selection list box that queries its CSetOfConnectableDatasets object, generates a list of data link names, invokes a general selection list box, and returns the selected dataset name.

CEditorWindow: A general purpose read/write window class for viewing and editing an ASCII text file. No attempt is made to assess the validity or correctness of the edited information. This class will encapsulate the UIMX editor dialog form.

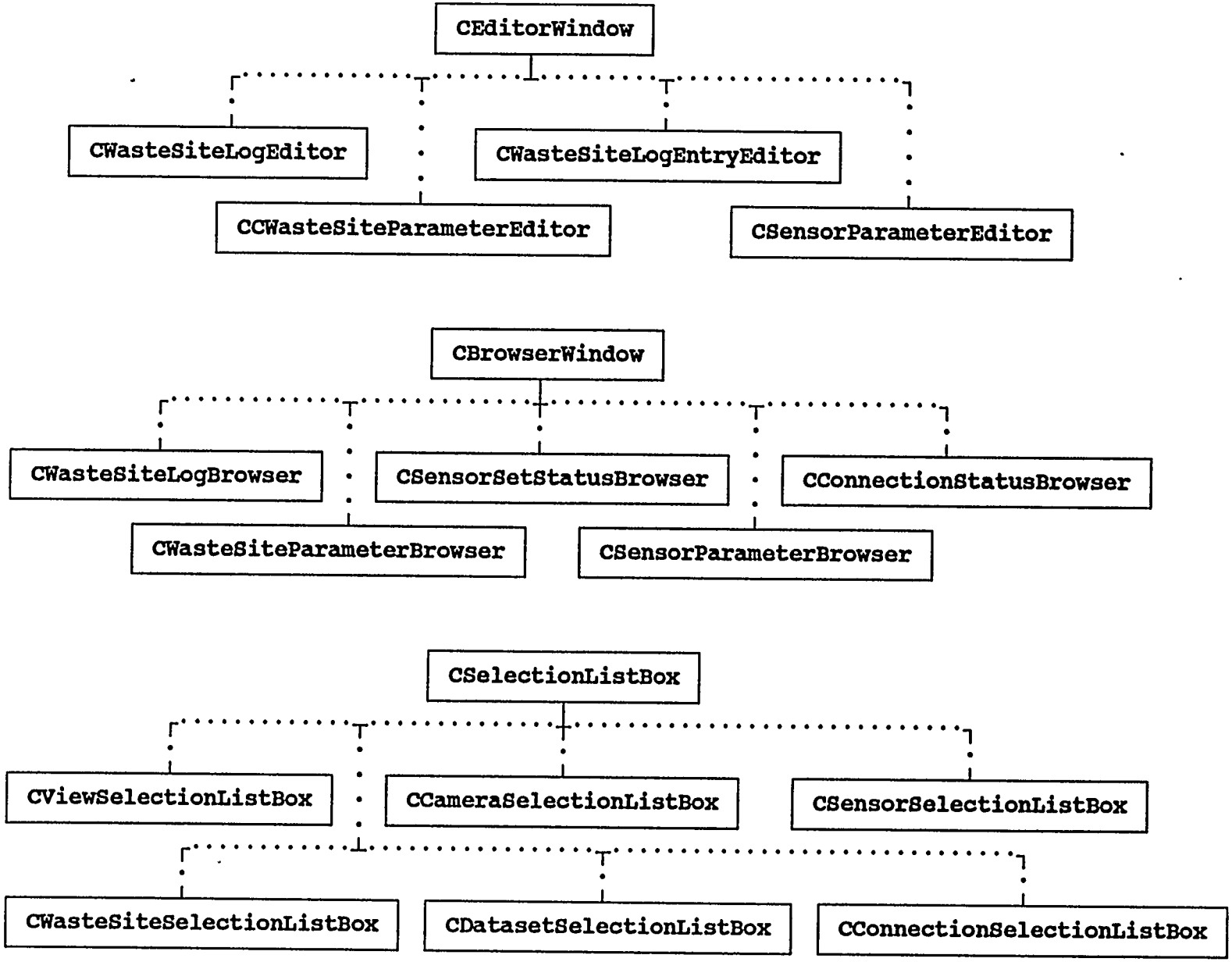


Figure 6-10: ICERVS Phase II Demonstration CSCI
OOD Diagram #4

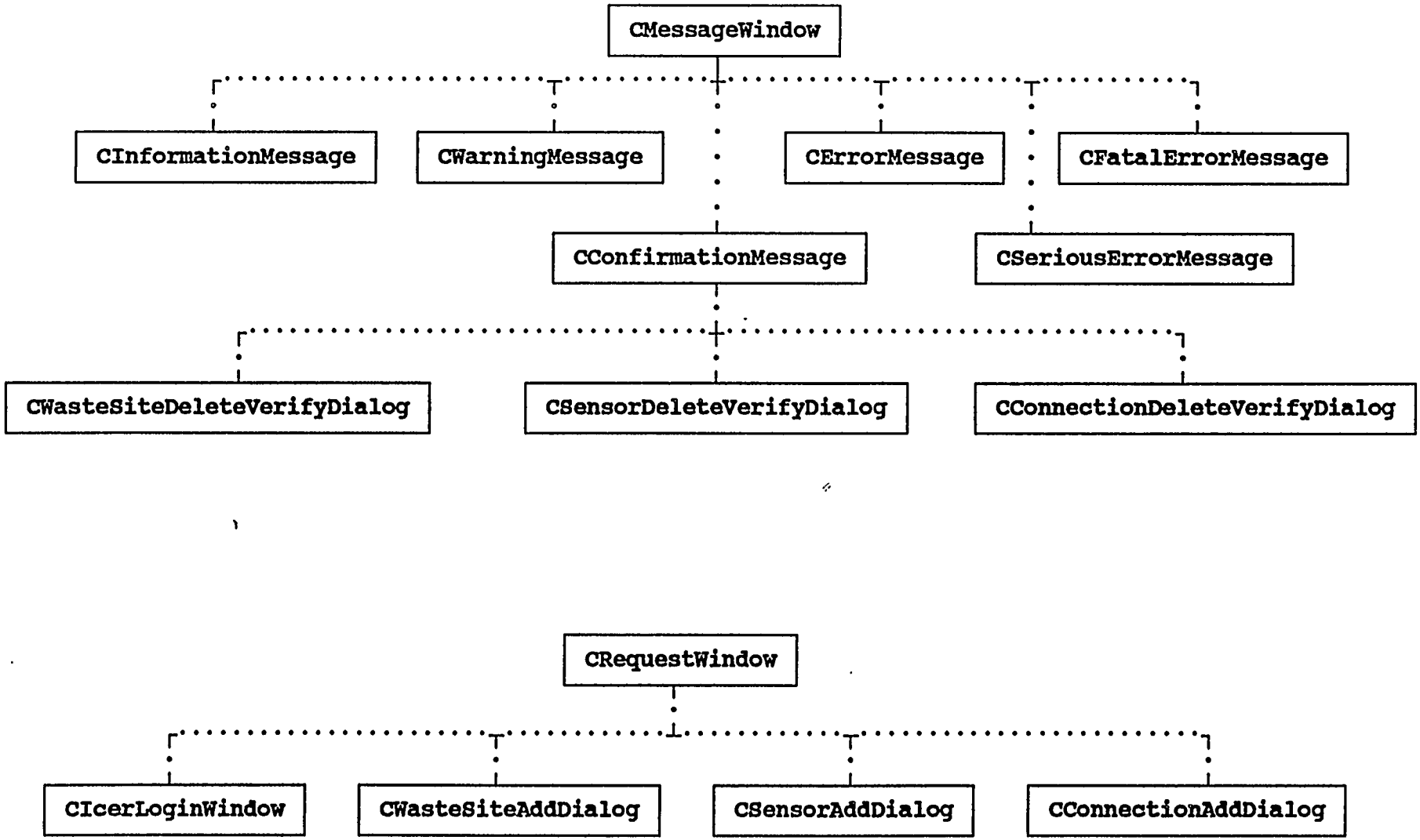
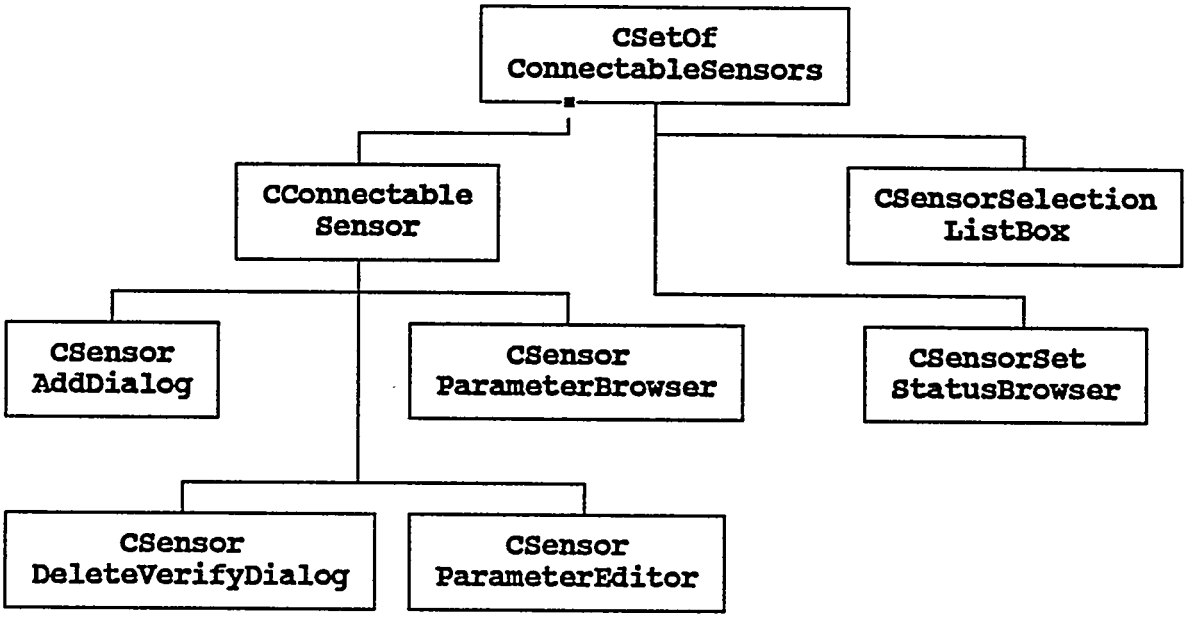


Figure 6-11: ICERVS Phase II Demonstration
OOD Diagram #5



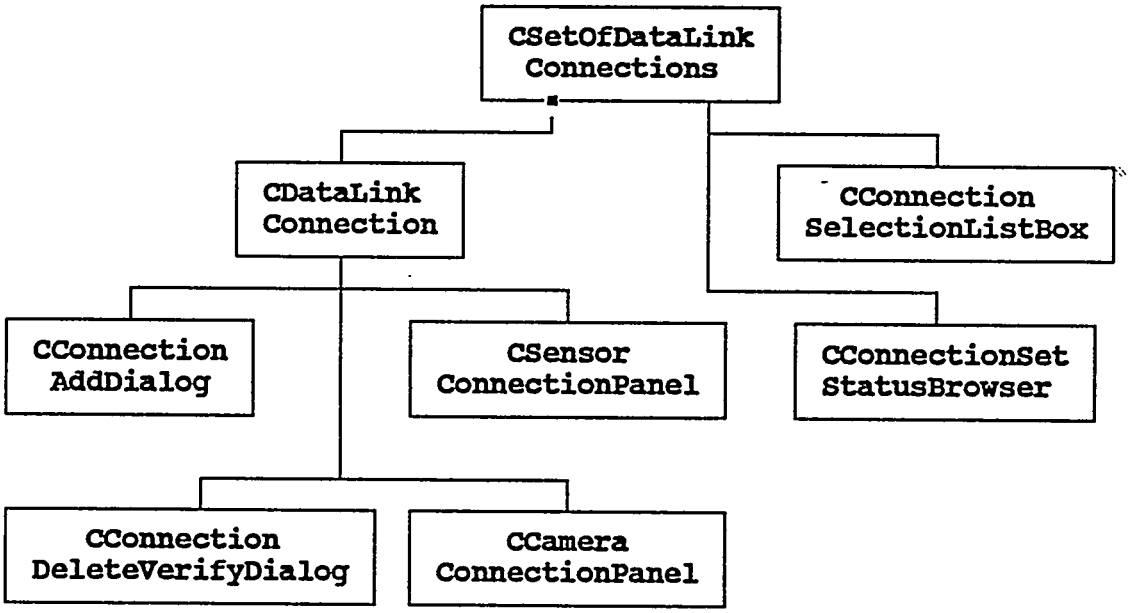


Figure 6-12: ICERVS Phase II Demonstration CSCI
OOD Diagram #6

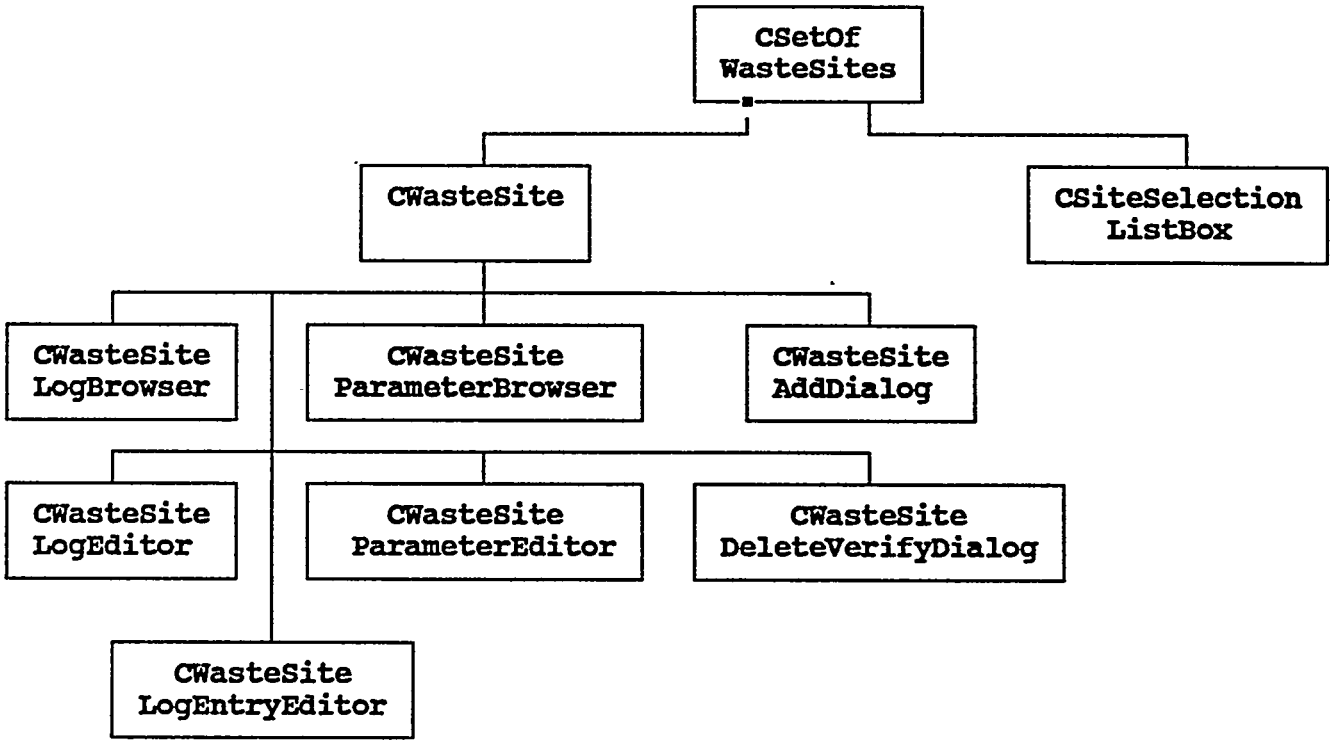


Figure 6-13: ICERVS Phase II Demonstration CSCI OOD Diagram #7

CFileSelectionListBox: A specialized selection list class that presents a list of filenames in a window and allows the operator to select one or more entries. This class will encapsulate the UIMX file selection dialog.

CGlobalHumanInterfaceComponentData: A class that holds all necessary application dependent global HIC data parameters. All data members of this class are static.

CIcerHelpManager: The class that implements the ICERVS help system. This class is not implemented in Phase II. This class will encapsulate the UIMX help dialog form.

CIcerLoginWindow: A class that encapsulates all aspects of logging into the ICERVS system. An instance of this class is automatically created, used and destroyed during the start-up of the ICERVS software subsystem. Login may also be performed from the main window SYSTEM menu.

CIcerMainWindow: A class that implements the ICERVS main window and its menus. Exactly one instance of CIcerMainWindow exists while the ICERVS software subsystem is active. This class will encapsulate the UIMX main window.

CMessageWindow: A general purpose class for displaying a message in a window. One or more buttons may appear on the window depending upon the specific purpose of the message window. Classes **CInformationMessage, CConfirmationMessage, CWarningMessage, CErrorMessage,** and **CFatalErrorMessage** are derived from **CMessageWindow**. This class will encapsulate the UIMX message dialog.

CRequestWindow: A general purpose class for displaying one or more prompts and inputting the operator's response. This class will encapsulate the UIMX request dialog form.

CSelectionListBox: A general purpose selection list class that presents a list of ASCII text strings in a window and allows the operator to select one or more entries. This class will encapsulate the UIMX selection dialog.

CSensorAddDialog: A class that displays an input request dialog box to request a name and description for the newly created sensor.

CSensorConnectionControlPanel: A specialized window for specifying connections between sensors and volumetric databases. This class will encapsulate the UIMX sensor connect dialog form.

CSensorDeleteVerifyDialog: A class that display a verify dialog box to confirm the deletion of a sensor from a waste site.

CSensorParameterBrowser: A specialization of CBrowseWindow used to view/print the parameters for a sensor.

CSensorParameterEditor: A specialization of CBrowseWindow used to modify the parameters for a sensor.

CSensorSelectionListBox: A specialized selection list box that queries its SetOfConnectableSensors object, generates a list of sensor names, invokes a general selection list box, returns the selected sensor name.

CViewSelectionListBox: A specialized selection list box that queries its SetOfConnectableViews object, generates a list of data link names, invokes a general selection list box, and returns the selected view name.

CSensorSetStatusBrowser: A class that displays the set of sensors for a site and status information about the sensors.

CWasteSiteAddDialog: A class that displays an input request dialog box to request a name and description for the newly created waste site.

CWasteSiteDeleteVerifyDialog: A class that display a verify dialog box to confirm the deletion of a waste site.

CWasteSiteLogBrowser: A specialization of CBrowseWindow used to view/print a waste site log file.

CWasteSiteLogEditor: A specialization of CEditorWindow used to edit the site log file.

CWasteSiteLogEntryEditor: A specialization of CEditorWindow used to create or edit an single entry from the site log file.

CWasteSiteParametersBrowser: A specialization of CBrowseWindow used to view/print the parameters that define a waste site.

CWasteSiteParametersEditor: A specialization of CEditorWindow used to edit the parameters that define a waste site.

CWasteSiteSelectionListBox: A specialized selection list box that queries its CSetOfWasteSites object, generates a list of sites names, invokes a general selection list box, returns the selected site name.

6.5.2. Globally Available Data

Figure 6-9 introduced a PDC "super" class (CGlobalPDC) that contained the sets of waste sites, sensors, material properties, and connections. This class was invented to be a convenient repository of the above data (and other data such as operator name, system data path, etc.) which will be needed by many other PDC and HIC classes. Rather than make each of the data items global, it was decided to encapsulate the concept of global data into a single class and object. This class contains only static data members and static access functions for the data. Since the data members are static, all instances of the class share the same data members. This means that an object wishing to access the application global data need only instantiate a local object of the CGlobalPDC class. When completed, the local object is deleted. Thus, access to application global data is simple and requires minimal interaction.

6.5.3. Major Function Descriptions

SYSTEM MENU:

Login *(Login as ICERVS operator)*: Uses CIcerLoginWindow object to prompt for operator name via a CRequestWindow object and verify that the specified name is for a valid ICERVS operator. An instance of CGlobalPDC is created to store the operator name in the application global data store. The callback function is CIcerMainWindow::SystemLogin().

About: (*Display ABOUT ICERVS information*): Displays an ABOUT message for the ICERVS system by creating a CAboutDialog object. The callback function is CIcerMainWindow::SystemAboutIcervs().

Quit *(Exit ICERVS system)*: Closes all window, releases all resources and exits the ICERVS Phase II Demonstration CSCI. The callback function is CIcerMainWindow::SystemQuit().

SITE MENU

Select *(Select a current site)*: Uses a CWasteSiteSelectionListBox object to display a list of system waste site names and to allow the operator to select a site name. The name of the current site is saved in the global CSetOfWastes object contained in the CGlobalPDC object. The ICERVS main window is updated to reflect new current site and all connected servers are notified of the new current site. The callback function is CIcerMainWindow::SiteSelect().

Add *(Create a new site)*: Uses a CWasteSiteAddDialog object to prompt for a site name and description. Invokes CSetOfWasteSites::CreateNewWasteSite method to create a new site directory with default files, add an entry to the site log file and add the new site to the global CSetOfWasteSites object. A CWasteSiteParameterEditor is instantiated for the new site to all editing of the site parameters. All connected servers are notified of the new site. The callback function is CIcerMainWindow::SiteAdd().

Delete *(Delete an existing site)*: Uses a CWasteSiteSelectionListBox object to display a list of system waste site names and to allow the operator to select a site name. Queries all connected sensors to ensure that the selected site is not active. Removes the existing site directory and its files after operator verification via a CWasteSiteDeleteVerifyDialog object. All connected servers are notified of the deletion of the site. The callback function is CIcerMainWindow::SiteDelete().

Append Log *(Add a new entry to site log file)*: Uses a CWasteSiteSelectionListBox object to display a list of system waste site names and to allow the operator to select a site name. Creates a new CWasteLogEntry object and uses a CWasteSiteLogEntryEditor to allow the operator may enter any information desired. The information is then formatted into a log file entry and appended to the site log file by using the CWasteSite::AppendToLog method. The callback function is CIcerMainWindow::SiteAppendLog().

Edit Log *(Edit the site log file)*: Uses a CWasteSiteSelectionListBox object to display a list of system waste site names and to allow the operator to select a site name. Uses a CWasteSiteLogEditor to allow editing of a site log file via a scrollable, multi-line edit window. The callback function is CIcerMainWindow::SiteEditLog(). This function is not implemented in Phase II.

Print Log *(Print the site log file)*: Uses a CWasteSiteSelectionListBox object to display a list of system waste site names and to allow the operator to select a site name. Uses a CWasteSiteLogBrowser object to allows viewing and/or printing of the site log file. The callback function is CIcerMainWindow::SitePrintLog().

Edit Parameters *(Edit the site parameters file)*: Uses a CWasteSiteSelectionListBox object to display a list of system waste site names and to allow the operator to select a site name. Uses a CWasteSiteParameterEditor to allow editing of a site parameters file via a scrollable, multi-line edit window. After editing is complete, the modified site file is read to verify its parameters. All connected servers are notified of that parameters may have changed for the selected site. The callback function is CIcerMainWindow::SiteEditParameters().

Print Parameters *(Print the site parameters file)*: Uses a CWasteSiteSelectionListBox object to display a list of system waste site names and to allow the operator to select a site name. Uses a CWasteSiteParameterBrowser to allow viewing and/or printing of the a site parameters file. The callback function is CIcerMainWindow::SitePrintParameters().

VOLUMETRIC DATA MENU

Open *(Open Volumetric Data Window)*: Establishes a client-server link with the ICERVS Volumetric Data Server. If opened in interactive mode (the only mode supported for Phase II), the server creates a Volumetric Data Window on the client's display (the SGI) and allows

the operator to select an existing data set or create a new data set. Once a data set has been selected, various functions can be performed on the current data set. Figure 6-3 illustrates the Volumetric Data Window. Refer to paragraph 7.10 for details concerning the Volumetric Data Window. The callback function is CIcerMainWindow::VolumetricDataOpen().

CONNECTIONS MENU

Connect Sensor *(Connect a sensor to a data set)*: Queries all connected servers to generate a list of active site names and to construct a temporary CSetOfWasteSites object. Queries the appropriate server for each active site to determine the names of all active datasets for the site. Instantiates a CSensorConnectionControlPanel for the set of active sites. The operator is allowed to select the waste site, dataset, sensor, and material property data type. A CDataLinkConnection object is created and added to the global CSetOfDataLinkConnections object. Appropriate commands are issued to the sensor server and the volumetric data server to connect the sensor and the dataset. Once connected, data will flow from the sensor to the data set without any operator intervention. Figure 6-4 illustrates the sensor connection control panel. The callback function is CIcerMainWindow::ConnectionsConnectSensor().

Connect Camera *(Connect a camera to a data set)*: Queries all connected servers to generate a list of active site names and to construct a temporary CSetOfWasteSites object. Queries the appropriate server for each active site to determine the names of all active views for the site. Instantiates a CCameraConnectionControlPanel for the set of active sites. The operator is allowed to select the waste site, view, and camera. A CDataLinkConnection object is created and added to the global CSetOfDataLinkConnections object. Appropriate commands are issued to the camera server and the volumetric data server to connect the camera and the view. Once connected, the views on the data set can be commanded to track with the scene from the camera. Figure 6-5 illustrates the camera connection control panel. The callback function is CIcerMainWindow::ConnectionsConnectCamera().

Disconnect *(Disconnects a sensor or camera from a data set)*: Uses a CConnectionSelectionListBox to display a list of all current CDataLinkConnection names and to allow the operator to selected a connection name. Breaks the selected connection between a sensor/dataset or camera/view. Removes the CDataLinkConnection object from the global CSetOfDataLinkConnections object. The callback function is CIcerMainWindow::ConnectionDisconnect().

Information *(Display connection information)*: Uses a CConnectionSetStatusBrowser to display a window that shows information about the currently defined connections. The data may be optionally printed. The callback function is CIcerMainWindow::ConnectionInformation().

SENSOR MENU

Open *(Open a sensor for data acquisition)*: Uses a CWasteSiteSelectionListBox object to display a list of system waste site names and to allow the operator to select a site name. Uses a CSensorSelectionListBox to display a list of inactive sensors for the selected site and to allow the operator to select a sensor name to open. Locates the CConnectableSensor object in the CsetOfConnectableSensors for the site. Establishes a client-server link with the ICERVS Sensor Interface Server and the server for a selected sensor for a particular site. The sensor is declared active. If opened in interactive mode (the only mode supported for Phase II), the sensor server will open a control panel window on the client's display (the SGI). The sensor may then be configured and manipulated via its control panel. A sensor control panel may be closed without deactivating its associated sensor. Control panels for multiple sensors may be open on the client screen at the same time. Once the client-server link has been established, the client application may receive sensor data. The callback function is CIcerMainWindow::SensorOpen().

Close *(Close currently open sensor)*: Uses a CWasteSiteSelectionListBox object to display a list of system waste site names and to allow the operator to select a site name. Uses a CSensorSelectionListBox to display a list of active sensors for the selected site and to allow the operator to select a sensor name to close. Locates the CConnectableSensor object in the CsetOfConnectableSensors for the site. Dissolves the client-server link with the selected sensor. The sensor control panel will be closed and the sensor will become inactive. Any data link connects must be dissolved first. The callback function is CIcerMainWindow::SensorClose().

Add *(Add a new sensor to a site)*: Uses a CWasteSiteSelectionListBox object to display a list of system waste site names and to allow the operator to select a site name. Uses a CSensorSelectionListBox to display a list of system-wide sensor types and to allow the operator to select a sensor type to add to the selected site. Uses a CSensorAddDialog to prompt the operator for a name and description for the new sensor. Adds the new CConnectableSensor object to the CSetOfConnectableSensors for the site. Creates a CSensorParameterEditor to allow tailoring of the sensor parameters. Notifies all connected servers that a new sensor has been added to the site. The callback function is CIcerMainWindow::SensorAdd().

Delete *(Delete sensor from a site)*: Uses a CWasteSiteSelectionListBox object to display a list of system waste site names and to allow the operator to select a site name. Uses a CSensorSelectionListBox to display a list of all sensors for the selected site and to allow the operator to select a sensor name to delete. Locates the CConnectableSensor object in the CsetOfConnectableSensors for the site. Verifies that the sensor is not connected to a dataset. Uses a CSensorDeleteVerifyDialog to confirm deletion of the sensor. Automatically closes the sensor if it is active and then deletes the CConnectableSensor object from the

CSetOfConnectableSensors for the site. Notifies all connected servers that the sensor has been deleted from the site. The callback function is CIcerMainWindow::SensorDelete().

Edit (Edit site specific sensor parameters file): Uses a CWasteSiteSelectionListBox object to display a list of system waste site names and to allow the operator to select a site name. Uses a CSensorSelectionListBox to display a list of active sensors for the selected site and to allow the operator to select a sensor name to close. Locates the CConnectableSensor object in the CSetOfConnectableSensors for the site. Creates a CSensorParameterEditor to allow editing of the sensor parameter values. Upon completion of the editor, reads the sensor parameter file to verify the parameters. Notifies all connected servers that the parameters for the sensor may have changed. The callback function is CIcerMainWindow::SensorEditParameters().

Print (Print site specific sensor parameters file): Uses a CWasteSiteSelectionListBox object to display a list of system waste site names and to allow the operator to select a site name. Uses a CSensorSelectionListBox to display a list of active sensors for the selected site and to allow the operator to select a sensor name to close. Locates the CConnectableSensor object in the CsetOfConnectableSensors for the site. Creates a CSensorParameterBrowser to allow viewing and/or printing of the sensor parameter file. The callback function is CIcerMainWindow::SensorPrintParameters().

Status *(Display information about sensors)*: Uses a CWasteSiteSelectionListBox object to display a list of system waste site names and to allow the operator to select a site name. Uses a CSensorSetStatusBrowser to display a window that shows information about the currently defined sensors for the site. The data may be optionally printed. The callback function is CIcerMainWindow::SensorStatus().

HELP MENU

Index: (*Display HELP index*): This function displays an index of HELP information in a list selection dialog window. When the operator selects the desired HELP index item, all information about that item will be displayed in a scrollable multi-line browse dialog window. The callback function is CIcerMainWindow::HelpIndex(). This function is not implemented in Phase II.

Extended: (*Display extended HELP for the system*): This function displays a scrollable multi-line browse dialog window containing system wide HELP information. The callback function is CIcerMainWindow::HelpExtended(). Minimal help information is available for Phase II.

7. VOLUMETRIC DATA SUBSYSTEM CSCI DESIGN

The Volumetric Data CSCI encapsulates the storage, retrieval, manipulation, and visualization of spatial, property, and geometric object data.. Six of the seven CSCs (excepting the Visualization and Interaction CSC) form the core of the ICERVS Volumetric Data Subsystem Server (VDS Server). The Visualization and Interaction CSC acts as a client of the ICERVS Volumetric Data Server. Figure 7-1 (duplicate of Figure 4-6) illustrates the CSCs for this CSCI. The sections that follow detail the design of the software for this CSCI.

7.1. General Design Approach

The general design of this CSCI is that of a server process that requires connection with one or more (one for Phase II) UNIX-based client application processes. These clients may reside on the same system as the server or different systems. (For Phase II, the clients and servers will reside on the same system.) UNIX sockets, using Internet Protocol (IP), and Sun RPC will be used for communications. In addition, the Sandia developed GENISAS and ISOE software, which is based on sockets, will be used to provide the low-level client-server support. Section 5.0 of this document details the client-server communications design and the classes defined in that section will be used to facilitate the implementation of the CSCI main program and the Volumetric Client Data Interface CSC.

The analysis and design of this CSCI (as well as for all the other CSCIs) begins with the identification of the major Problem Domain Component (PDC) classes. An analysis of the requirements for this CSCI readily leads to the identification of the following major PDC objects (i.e. software classes).

Site	Geometric Object
Dataset	Property Data
View	Dimensional Data
Cutplane	SensorData

The relationships among these classes is illustrated in the OOA diagram of Figure 7-2. The Volumetric Data Subsystem is concerned primarily with **Dataset** objects. A dataset is associated with a site (underground storage tank, buried waste pit, etc.). One or more **View** objects may be defined to visualize the dataset. The dataset itself contains several types of data objects: **Geometric Object, Dimensional Data, Property Data** and **Sensor Data**. One or more **Cutplane** objects may be defined on the dataset to slice or bound the information presented by the view object. Figure 7-3 provides another view of the class interactions for the Volumetric Data CSCI.

7.2. Requirements Allocation

Allocation of the ICERVS system requirements to this CSCI was given in Table 4-1. This section will refine that allocation to the CSC level. The Volumetric Data Client Interface CSC does not derive from any specific ICERVS system requirement(s). Its inclusion in the Volumetric Data CSCI was a design choice that followed from the architectural decision to embrace a client-server architecture. The requirements for the remaining Volumetric Data Subsystem CSCs are summarized in Tables 7-1 through 7-6.

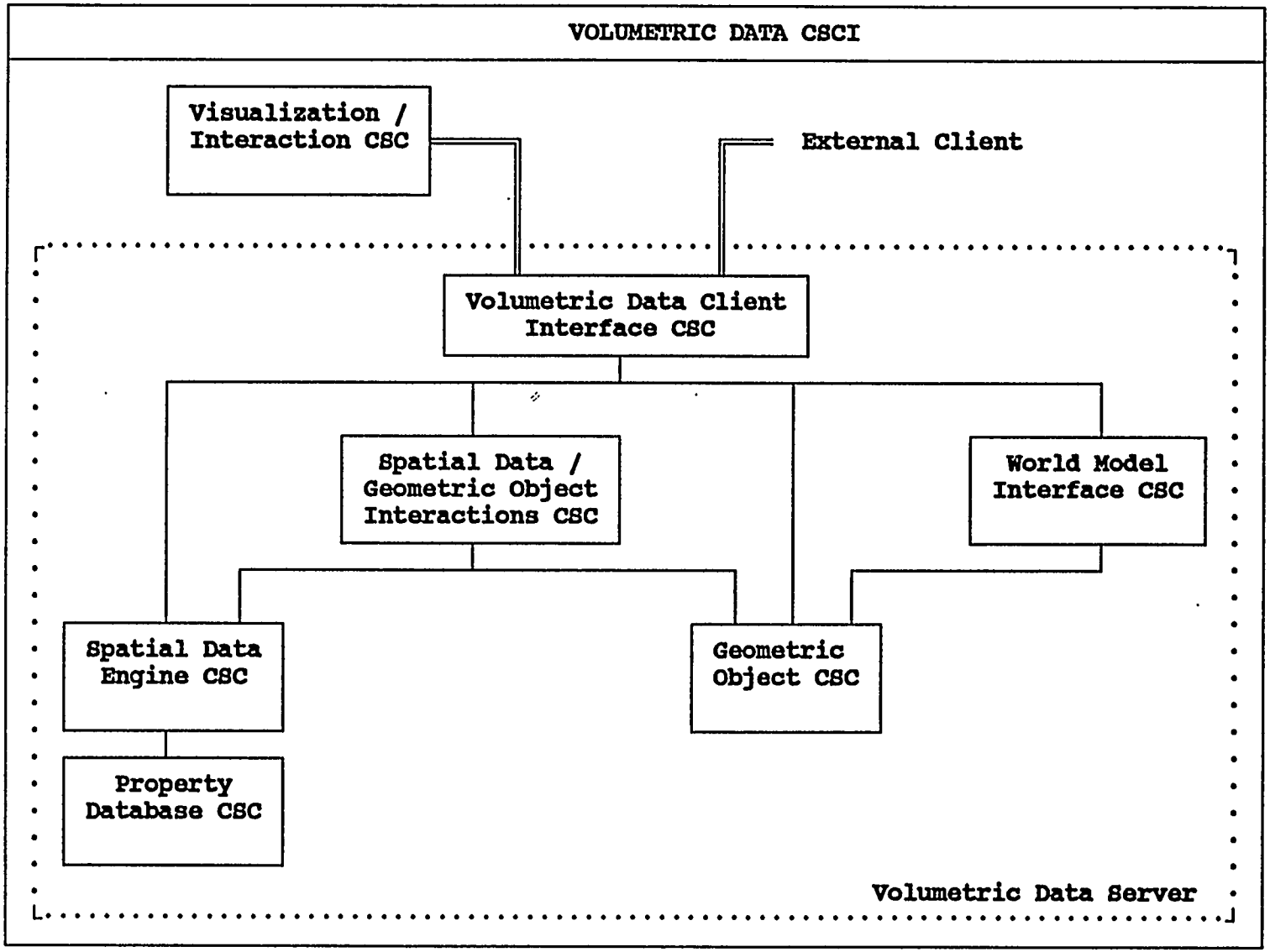


Figure 7-1: Volumetric Data CSCI Block Diagram

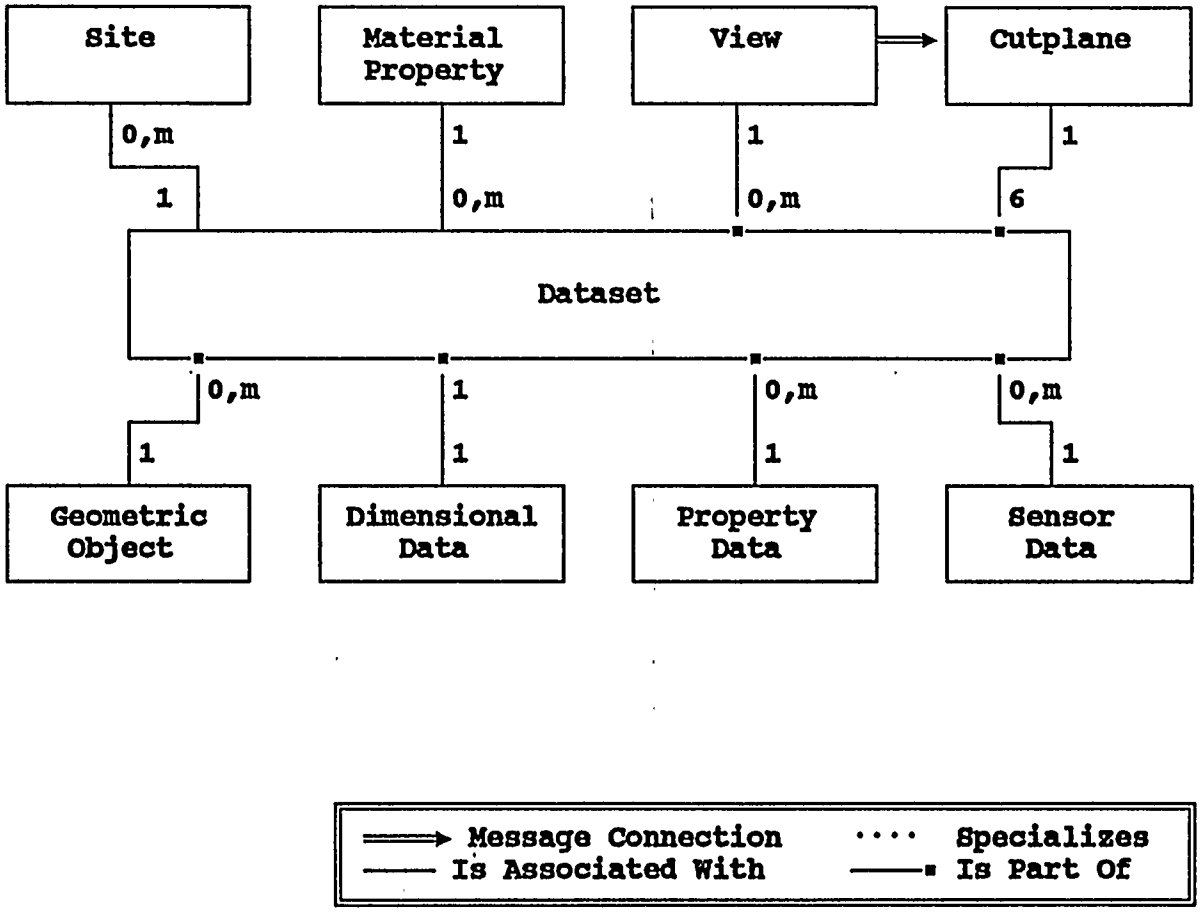


Figure 7-2 Volumetric Data Subsystem OOA Diagram

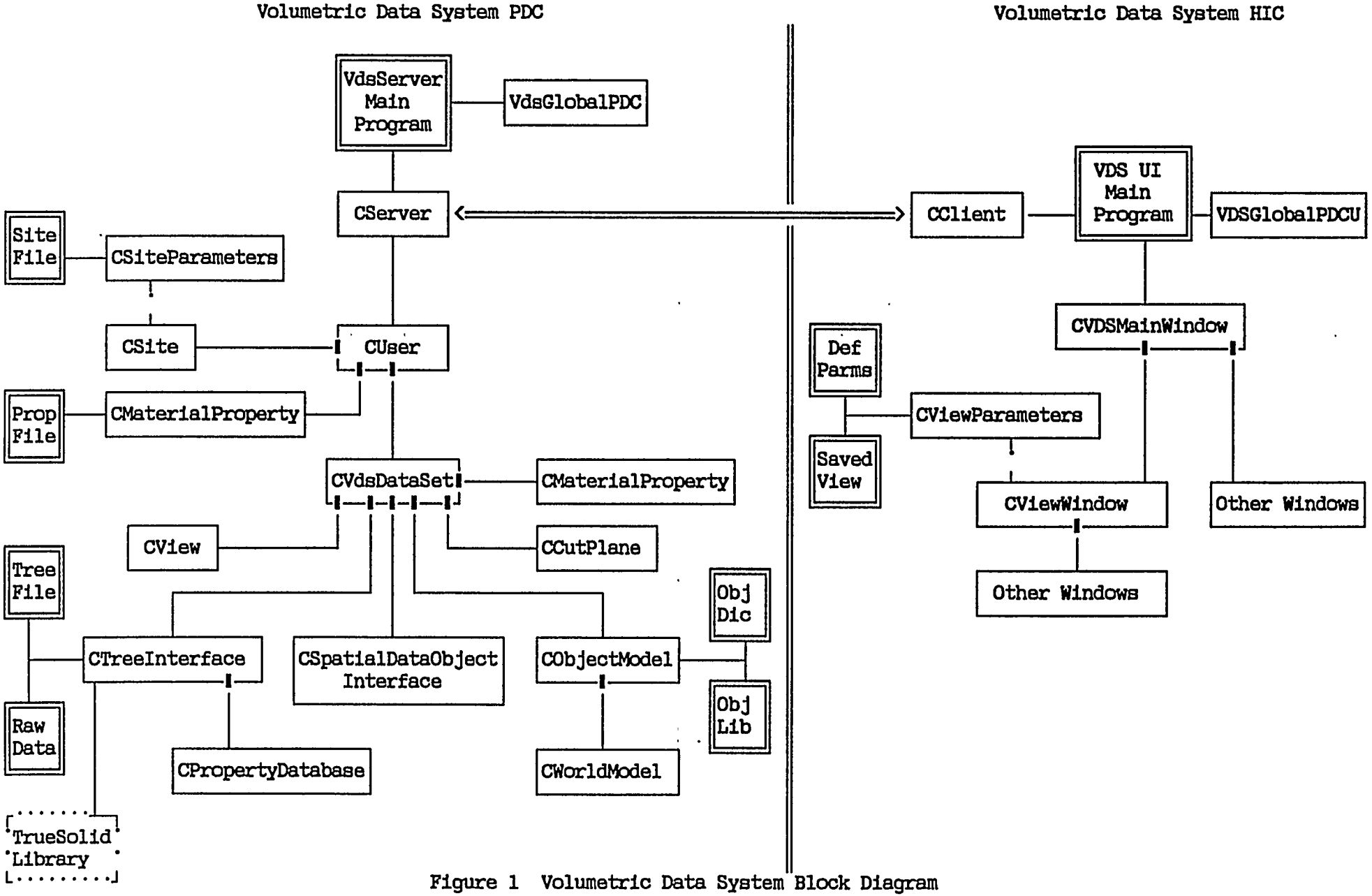


Figure 1 Volumetric Data System Block Diagram

Table 7-1 Requirements For Spatial Data Engine CSC

Requirement Number	Description
R1.01	Octree: spatial data
R1.02	Octree: property data
R1.03	Octree: spatial interpolation
R1.04	Octree: linear resolution 1:512, expandable
R1.10	Octree: sensor data
R5.01	Copy octree
R5.02	Set region within tree to selected state
R5.06	Compare two octrees, compute difference
R5.07	Compute 2.5D surface map from octree
R5.08	Compute difference 2.5D surface map
R5.09	*Surface connectivity*
R6.02	Save / Retrieve waste site datasets to/from disk
R6.03	Build octree from backup raw data
R6.05	Multiple system of units

Table 7-2 Requirements For Geometric Object CSC

Requirement Number	Description
R1.05	Geom: polyhedral objects
R1.06	Geom: geometric primitives
R1.07	Geom: associated text each object
R1.08	Geom: 100 objects, expandable
R2.01	Library of primitives / templates
R2.02	Standard templates
R2.03	User-defined templates
R2.05	Synthesize 2D polygons
R2.06	Synthesize 3D polyhedra
R2.08	Attach text to objects
R5.03	Operator delete objects
R6.02	Save / Retrieve waste site datasets to/from disk
R6.05	Multiple system of units
R6.06	*Define Disassembly data*

Table 7-3 Requirements For Spatial Data / Object Interactions CSC

Requirement Number	Description
R2.04	*Automatic waste surface modeling*
R5.04	**Scan object for consistency with octree**
R5.05	**Compare octree and object data**

Table 7-4 Requirements For World Model Data Interface CSC

Requirement Number	Description
R1.09	*Geom: enter architectural and robot plans*
R7.06	**Output: geometric model data**
R7.07	**Output: 2/5D surface map data**

Table 7-5 Requirements For Property Database CSC

Requirement Number	Description
R1.02	**Octree: property data**
R1.11	**Octree: property data interpolation**
R5.11	**Combine multiple property data**
R6.02	**Save / Retrieve waste site datasets to/from disk**

Table 7-6 Requirements For Visualization/Interaction CSC

Requirement Number	Description
R2.07	Dimensioning tools
R3.01	Translation and scaling
R3.02	Display coordinate axes
R3.03	Parallel cut planes
R3.04	Display object text data
R3.05	Shaded or wire frame object display
R3.06	Update octree display as points received
R3.07	Pseudo-color octree data
R3.08	Pseudo-color geometric objects
R3.09	Text display view parameters
R3.10	Save / Recall view parameter set
R3.11	Multiple windows displaying same data
R3.13	Display 2.5D surface map
R3.14	*Display views of spatial and property data*
R6.05	Multiple system of units

For a more detailed description of each requirement, refer to Appendix D of the *ICERVS Phase II System Design Report.* In the tables, requirements that apply (full or partial implementation) to ICERVS Phase II, including those from Phase I, are in **bold** type. Other requirements that are not part of Phase II but strongly influence the software design are *italicized.* In some cases requirements span more than one CSC and will appear in more than one table.

7.3. Volumetric Data Client Interface CSC Detailed Design

This CSC is responsible for accepting commands from client applications, parsing the command messages, validating the command, and dispatching the command to the command execution functions. These execution functions will typically call upon routines in the other CSCs that comprise the Volumetric Data CSCI. The software contained in this CSC follows the general approach described in section 5.0 for client-server implementations. Three specialized software classes are used: CVolumetricDataClient, CVolumetricDataServer, and CVolumetricDataUser.

The **CVolumetricDataClient** class provides the client process (Demonstration Application CSC or VDS Visualization and Interaction CSC) with a standard interface to the Volumetric Data Server. This class provides methods (functions) that parallel the command functions of the VdsServer. The details of the client side of the client-server interface are completely encapsulated in the CVolumetricDataClient class.

The **CVolumetricDataServer** class provides the server process (VdsServer) with a standard communications interface with the client application and implements several commands related to managing the client interface (connect, disconnect, list of users, etc.). Most command functions are passed on to the CVolumetricDataUser class for execution.

Objects of the **CVolumetricDataUser** class represent the client in the server process. Commands are received from the CVolumetricDataServer object and executed by one of the CVolumetricDataUser methods. Each command execution function formats a reply message and returns that reply to the CVolumetricDataServer for transmission to the client. At present, one CVolumetricDataUser object will exist, but in the future, it is envisioned that the VdsServer will become multi-cliented.

7.3.1. VDS Server Commands

The VDS Server supports a number of command functions which are organized as followings:

1. General Commands
2. Site Commands
3. Spatial Data Commands
4. Property Commands
5. Geometric Object Commands

Tables 7-7 through 7-11 identify the commands in each category and provide a brief description of each of the commands. The general command/reply syntax is also given.

Table 7-7 VDS General Commands

ConnectToServer: Identifies a new VDS user. If accepted, the user is added to the user list. Returns a userId for the new VDS user. Command: ConnectToServer userName dataPath Reply: ConnectToServer status userId
DisconnectFromServer: Closes VDS connection to the user and removes the user from the user list. All user datasets will be closed, but not saved. Command: DisconnectFromServer userId Reply: DisconnectFromServer status
EnableUserInterface: Displays standard VDS operator interface on specified display screen Command: EnableUserInterface userId displayName Reply: EnableUserInterface status
GetListOfUsers: Returns a list of VDS user ids and names. Command: GetListOfUsers Reply: GetListOfUsers status numUsers <id name description> ....
GetServerCommandList: Returns a list of server command names, number of arguments and argument names for each command. Command: GetServerCommandList Reply: GetServerCommandList status numCmd <name numArg arg1 ...> ...
GetUserId: Returns user id associated with the given user name. If multiple occurrences of the same name exist, the id associated with the first occurrence is returned. Command: GetUserId userName Reply: GetUserId status userId
GetUserName: Returns the user name for the given user id. Command: GetUserName userId Reply: GetUserName status userName
SetDataPath: Defines the datapath associated with the given user id. Command: SetDataPath userId dataPath Reply: SetDataPath status

Table 7-7 VDS General Commands (continued)

SetOperatorName:	Defines the ICERVS operator name associated with the given user id.
Command:	SetOperatorName userId operatorName
Reply:	SetOperatorName status
SetUserName:	Defines the name associated with a given user id.
Command:	SetUserName userId userName
Reply:	SetUserName status
ShutdownServer:	Causes VDS server process to perform an orderly shutdown.
Command:	ShutdownServer
Reply:	ShutdownServer status

Table 7-8 VDS Site Commands

CreateNewSite: Creates a new site directory and site files. Adds site to site list. Returns a site id. The siteId may be previously obtained from the RequestSiteId command. If blank, a new id will be assigned. Use the SetSiteParameter command to tailor the site parameters. Command: CreateNewSite userId siteName siteDescription siteId Reply: CreateNewSite status siteId
DeleteSite: Deletes a site from list of sites, deletes all site files, and removes the site directory. Command: DeleteSite userId siteId Reply: DeleteSite status
GetCurrentSite: Returns users current default site. Command: GetCurrentSite userId Reply: GetCurrentSite status siteId
GetListOfSites: Returns a list of site names for the user Command: GetListOfSites userId Reply: GetListOfSites status numSites <siteId siteName description> ....
GetSiteDataPath: Returns the datapath associated with the given site id. Command: GetSiteDataPath userId siteId Reply: GetSiteDataPath status dataPath
GetSiteId: Returns site id associated with the given site name. If multiple occurrences of the same name exist, the id associated with the first occurrence is returned. Command: GetSiteId userId siteName Reply: GetSiteId status siteId
GetSiteName: Returns the site name for the given site id. Command: GetSiteName userId siteId Reply: GetSiteName status siteName
GetSiteParameter: Returns value for specified site parameter Command: GetSiteParameter userId siteId prmId Reply: GetSiteParameter status prmValue
RequestSiteId: Allocates and returns a new site id that can be used in a CreateNew Site command. Command: RequestSiteId userId Reply: RequestSiteId status siteId

Table 7-8 VDS Site Commands (continued)

SetCurrentSite: Identifies the specified site as the users current default site. Command: SetCurrentSite userId siteId Reply: SetCurrentSite status
SetSiteParameter: Defines value for specified site parameter. Command: SetSiteParameter userId siteId prmId prmValue Reply: SetSiteParameter status
SiteAdded: Notifies VDS that a new site has been added to ICERVS system. Command: SiteAdded userId siteId Reply: SiteAdded status
SiteDeleted: Notifies VDS that a site has been deleted from ICERVS system. Command: SiteDeleted userId siteId Reply: SiteDeleted status
SiteParametersChanged: Notifies VDS that the parameters for the specified site may have been changed. Command: SiteParametersChanged userId siteId Reply: SiteParametersChanged status

Table 7-9a VDS Spatial Data Commands For Datasets

AddArrayOfPoints:	Adds an array of points to dataset.
Command:	AddArrayOfPoints userId datasetId numPoints dataPoint1 dataPoint2 ...
Reply:	AddArrayOfPoints status
AddFile:	Adds a file of points to the dataset.
Command:	AddFile userId datasetId fileName
Reply:	AddFile status
AddPoint:	Adds a single point to dataset.
Command:	AddPoint userId datasetId dataPoint
Reply:	AddPoint status
CloseAllDatasets:	Closes all datasets and deletes all data from memory. Does not save dataset contents prior to closing. If siteId is given as <ALL>, all datasets for all sites will be closed.
Command:	CloseAllDatasets userId siteId
Reply:	CloseAllDatasets status
CloseDataset:	Closes the specified dataset and deletes all data from memory. Does not save dataset prior to closing.
Command:	CloseDataset userId datasetId
Reply:	CloseDataset status
DeleteDataset:	Deletes the specified dataset and all data files associated with the dataset. The dataset must be open. All views on the dataset will be closed prior to deleting the dataset.
Command:	DeleteDataset userId datasetId
Reply:	DeleteDataset status
ErasePoint:	Erases a point from dataset.
Command:	ErasePoint userId datasetId dataPoint
Reply:	ErasePoint status
GetCurrentDataset:	Returns the current dataset.
Command:	GetCurrentDataset userId
Reply:	GetCurrentDataset status datasetId

Table 7-9a VDS Spatial Data Commands For Datasets (continued)

GetDatasetId: Returns dataset id for named dataset. If multiple occurrences of the same name exist, the id associated with the first occurrence is returned. Command: GetDatasetId userId datasetName Reply: GetDatasetId status datasetId
GetDatasetName: Returns dataset name for specified dataset. Command: GetDatasetName userId datasetId Reply: GetDatasetName status datasetName
GetDatasetParameter: Returns value for a dataset parameter. Command: GetDatasetParameter userId datasetId prmId Reply: GetDatasetParameter status prmValue
GetDatasetPath: Returns the data path for the specified dataset. Command: GetDatasetPath userId datasetId Reply: GetDatasetPath status dataPath
GetListOfAllDatasets: Returns a list of all dataset names for the specified user and/or site. If siteId = <ALL>, all datasets across all sites are returned. Otherwise, only the datasets for the specified site are returned. Command: GetListOfAllDatasets userId siteId Reply: GetListOfAllDatasets status numDatasets <id name description> ....
GetListOfOpenDatasets: Returns a list of open dataset names for the specified user. Command: GetListOfOpenDatasets userId Reply: GetListOfOpenDatasets status numDatasets <id name description> ....
NewDataset: Creates a new dataset for the given site. Creates all necessary data files. Returns a dataset id. The datasetId may be previously obtained from the RequestDatasetId command. If blank, a new id will be assigned. Command: NewDataset userId siteId datasetName description datasetId Reply: NewDataset status datasetId
OpenDataset: Opens an existing dataset for the given site. Does not read data into memory. Returns a dataset id. The datasetId may be previously obtained from the RequestDatasetId command. If blank, a new id will be assigned. Command: OpenDataset userId siteId datasetName datasetId Reply: OpenDataset status datasetId

Table 7-9a VDS Spatial Data Commands For Datasets (continued)

DatasetVolumetricDifference: Computes volumetric difference between two datasets. WHAT IS THE OUTPUT? Command: DatasetVolumetricDifference userId datasetId1 datasetId2 ????? Reply: DatasetVolumetricDifference status ?????
DatasetSurfaceMap: Computes surface map for one or two datasets. WHAT IS THE OUTPUT? Command: DatasetSurfaceMap userId datasetId1 datasetId2 ?????? Reply: DatasetSurfaceMap status ?????
ReadDataSet: Reads data from disk for specified dataset. Command: ReadDataSet userId datasetId Reply: ReadDataSet status
RequestDatasetId: Allocates and returns a new dataset id that can be used in a NewDataset or Open Dataset command. Command: RequestDatasetId userId siteId Reply: RequestDatasetId status datasetId
SaveAsDataset: Creates a new dataset and saves all data associated with the specified dataset in the new dataset. Returns a dataset id for the new dataset. Command: SaveAsDataset userId datasetId newSiteId newDatasetName Reply: SaveAsDataset status newDatasetId
SaveDataset: Saves all data associated with the specified dataset. Command: SaveDataset userId datasetId Reply: SaveDataset status
SetCurrentDataset: Declares specified dataset to be the current dataset. Command: SetCurrentDataset userId datasetId Reply: SetCurrentDataset status
SetDatasetParameter: Defines a dataset parameter value. Command: SetDatasetParameter userId datasetId prmId prmValue Reply: SetDatasetParameter status
SetRegion: Set all nodes of region to specified node type. Command: SetRegion userId datasetId *<<other parameters>>* Reply: SetRegion status
WriteDataset: Write data to disk for specified dataset. Command: WriteDataset userId datasetId Reply: WriteDataset status

Table 7-9b VDS Spatial Data Commands For Cutplanes

DeleteCutplane: Deletes a cutplane and removes it from the list of cutplanes for the associated dataset. Not Implemented. Command: DeleteCutplane userId cutplaneId Reply: DeleteCutplane status
GetCutplaneId: Returns id for specified cutplane name. If multiple occurrences of the same name exist, the id associated with the first occurrence is returned. Command: GetCutplaneId userId datasetId cutplaneName Reply: GetCutplaneId status cutplaneId
GetCutplaneName: Returns name for specified cutplane. Command: GetCutplaneName userId cutplaneId Reply: GetCutplaneName status cutplaneName
GetCutplaneParameter: Returns a cutplane parameter value. Command: GetCutplaneParameter userId cutplaneId prmId Reply: GetCutplaneParameter status prmValue
GetListOfCutplanes: Returns a list of cutplane ids, names, and descriptions for the specified dataset. Command: GetListOfCutplanes userId datasetId Reply: GetListOfCutplanes status numPlanes <id name description>....
NewCutplane: Creates a new cutplane and adds to set of cutplanes. Returns a cutplane id. The cutplaneId may be previously obtained from the RequestCutplaneId command. If blank, a new id will be assigned Not implemented. Command: NewCutplane userId datasetId cutplaneName description cutplaneId Reply: NewCutplane status cutplaneId
RequestCutplanetId: Allocates and returns a new cutplane id that can be used in a NewCutplane command.. Not implemented. Command: RequestCutplaneId userId siteId Reply: RequestCutplaneId status datasetId
SetCutplaneParameter: Defines a cutplane parameter value. Command: SetCutplaneParameter userId cutplaneId prmId prmValue Reply: SetCutplaneParameter status

Table 7-9c VDS Spatial Data Commands For Views

CloseAllViews: Closes all views for the specified dataset. If datasetId is given as <ALL>, all views for all datasets currently open by the specified user will be closed. Command: CloseAllViews userId datasetId Reply: CloseAllViews status
CloseView: Closes the specified view of a dataset. The view will be deleted from the view list. Command: CloseView userId viewId Reply: CloseView status
GetDefaultViewPrmFilename: Returns name of Default View Parameters File for the specified dataset. Command: GetDefaultViewPrmFilename userId datasetId Reply: GetDefaultViewPrmFilename status fileName
GetListOfViews: Returns a list of view ids and names for the specified dataset If dataset = <ALL>, returns list of all views across all datasets for specified user. Command: GetListOfViews userId datasetId Reply: GetListOfViews status numViews <viewId viewName description> ....
GetViewDisplayData: Returns a set of data to display for the specified view. Command: GetViewDisplayData userId viewId ????? Reply: GetViewDisplayData status ?????
GetViewId: Returns id for specified view name. If multiple occurrences of the same name exist, the id associated with the first occurrence is returned. Command: GetViewId userId datasetId viewName Reply: GetViewId status viewId
GetViewName: Returns view name for specified view id. Command: GetViewName userId viewId Reply: GetViewName status viewName
GetViewParameter: Returns value for specified view parameter. Command: GetViewParameter userId viewId prmId Reply: GetViewParameter status prmValue

Table 7-9c VDS Spatial Data Commands For Views (continued)

OpenView: Creates a new view for the specified dataset and adds to view list. Returns a new view id. The viewId may be previously obtained from the RequestViewId command. If blank, a new id will be assigned. Command: OpenView userId datasetId viewName description viewId Reply: OpenView status viewId
RequestViewId: Allocates and returns a new view id that can be used in an OpenView command.. Command: RequestViewId userId datasetId Reply: RequestViewId status viewId
SetViewParameter: Defines value for specified view parameter. Command: SetViewParameter userId viewId prmId prmValue Reply: SetViewParameter status

Table 7-10 VDS Property Data Commands

AddPropertyName: Adds new property name to the dataset. Returns property id. The propertyId may be previously obtained from the RequestPropertyId command. If blank, a new id will be assigned. Command: AddPropertyName userId datasetId propType propName description propId Reply: AddPropertyName status propertyId
DeleteProperty: Deletes property name from dataset. Command: DeleteProperty userId propertyId Reply: DeleteProperty status
GetListOfProperties: Returns a list of known property names for the specified dataset. Command: GetListOfProperties userId datasetId Reply: GetListOfProperties status numProp <id type name description>....
GetListOfPropertyTypes: Returns a list of known property types for the requesting client. Command: GetListOfPropertyTypes userId Reply: GetListOfPropertyTypes status numProp <id type description>....
GetPropertyId: Returns property id for specified property name. If multiple occurrences of the same name exist, the id associated with the first occurrence is returned. Command: GetPropertyId userId datasetId propertyName Reply: GetPropertyId status propertyId
GetPropertyName: Returns property name for specified property id. Command: GetPropertyName userId propertyId Reply: GetPropertyName status propertyName
GetPropertyParameter: Returns value for specified property parameter. Command: GetPropertyParameter userId propertyId prmId Reply: GetPropertyParameter status prmValue
RequestPropertyId: Allocates and returns a new property id that can be used in an AddPropertyName command. Command: RequestPropertyId userId datasetId Reply: RequestPropertyId status propertyId
SetPropertyParameter: Defines value for specified property parameter. Command: SetPropertyParameter userId propertyId prmId prmValue Reply: SetPropertyParameter status

Table 7-11 VDS Geometric Object Commands

CreateObject: Creates a new object and adds it to the dataset object list. Returns object id. The objectId may be previously obtained from the RequestObjectId command. If blank, a new id will be assigned. Object parameters must be defined with SetObjectParameter command. Command: CreateObject userId datasetId objectType objectName description objectId Reply: CreateObject status objectId
DeleteObject: Deletes specified object from object list. Command: DeleteObject userId objectId Reply: DeleteObject status
GetListOfObjects: Returns list of object ids, names, types and categories for specified object list. If datasetId = <ALL>, returns all objects across all datasets for the specified user. Command: GetListOfObjects userId datasetId Reply: GetListOfObjects status numObj <id name type description> ...
GetObjectId: Returns object id for specified object name. If multiple occurrences of the same name exist, the id associated with the first occurrence is returned. Command: GetObjectId userId datasetId objectName Reply: GetObjectId status objectId
GetObjectName: Returns object data for specified object. Command: GetObjectName userId objectId Reply: GetObjectName status objectName
GetObjectParameter: Returns specified parameter for specified object. Command: GetObjectParameter userId objectId prmId Reply: GetObjectParameter userId prmValue
ObjectVolumetricDifference: Performs volumetric difference between object and octree data contained within the object. WHAT IS THE OUTPUT? Command: ObjectVolumetricDifference userId datasetId objectId Reply: ObjectVolumetricDifference status ?????
ObjectConsistencyCheck: Performs object consistency check to confirm that every object contains some volumetric material. All list of inconsistent objects is returned. Command: ObjectConsistencyCheck userId datasetId objectId Reply: ObjectConsistencyCheck status numObjects <objectId> ...
RequestObjectId: Allocates and returns a new object id that can be used in a CreateObject command. Command: RequestObjectId userId datasetId Reply: RequestObjectId status objectId
SetObjectParameter: Defines specified parameter for the object in object list. Command: SetObjectParameter userId objectId prmId prmValue Reply: SetObjectParameter status

7.3.2 VDS Server Identifiers

As can be seen from the VDS Server command descriptions, several of the commands return identifiers that are required as arguments to many of the other commands. These identifiers are assigned by the VDS Server to uniquely identify dynamically created PDC objects (users, datasets, views, properties, etc.) that the VDS Server maintains on behalf of the client. The identifiers will be implemented as ASCII strings to promote transparency across interconnected UNIX systems. Client applications should NOT depend upon any specific implementation of identifiers. Each identifier should be viewed as unique and unrelated to any other identifier.

7.3.3. Class Descriptions

The Volumetric Data Client Interface CSC software is object oriented, implemented in C++ and consists of 10 classes. This section identifies the CSC classes and discusses their general characteristics (attributes, behavior and relationships). Section 7.3.4 will discuss the major functions assigned to the CSC and describe how the software classes implement the functions.

In keeping with good object-oriented analysis practices, the classes are grouped by the categories Problem Domain Component (PDC) and Human Interface Component (HIC).

7.3.3.1. PDC Classes

The Volumetric Data Subsystem OOA diagrams of Figure 7-2 and 7-3 have been expanded and specialized to produce the Volumetric Data Client Interface OOD diagrams in Figures 7-4, 7-5, and 7-6. Figure 7-4 has added three new classes (CVolumetricDataServer, CVolumetricDataClient and CVolumetricDataUser) to model the client-server interface. In addition, two previously defined classes (CMaterialProperty and CSite) have also been included in Figure 7-4. Figure 7-5 shows the derivation of the basic PDC classes and introduces a set of classes that form collections of the basic classes. (The RWCollectable and RWOrdered classes are from the Rogue Wave Tools++ library.) Figure 7-6 combines all the PDC classes and illustrates the Is A Part Of relationships among objects of the various classes.

The PDC classes for the Volumetric Data Client Interface are identified and described below.

CVdsDataset: A class that represents the data stored by ICERVS. The class is a part of the Spatial Data Engine CSC (see section 7.4), but is introduced here to illustrate the connection between the VDS Client Interface CSC and the Spatial Data Engine CSC.

CMaterialProperty: A class that represents the abstract idea of a material property (i.e., a temperature in degrees Fahrenheit). It does not include the material property data value (the value of the temperature). This class was previously defined in section 6.3 and is redefined here for completeness.

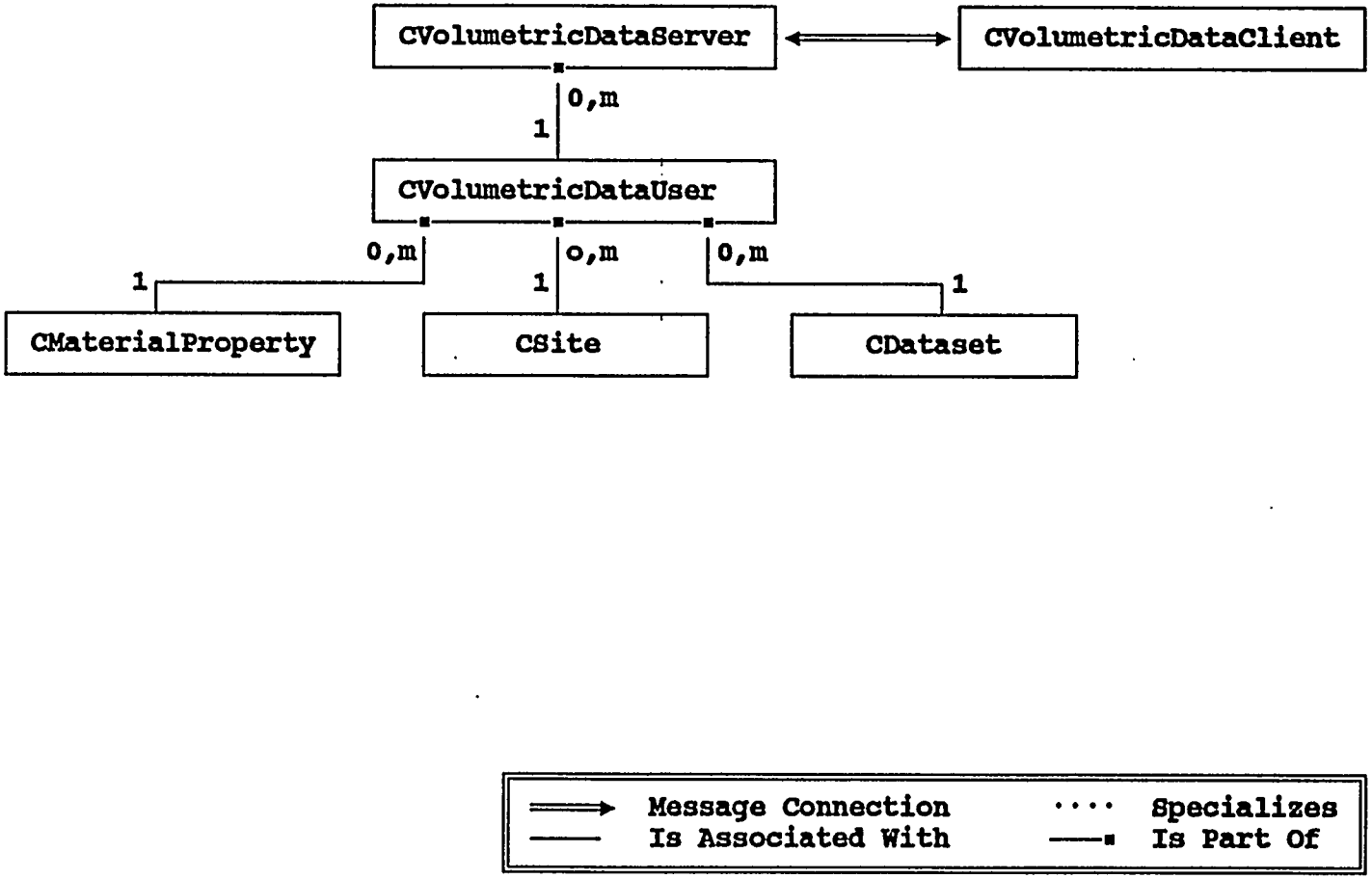


Figure 7-4 VDS Client Interface CSC OOA Diagram

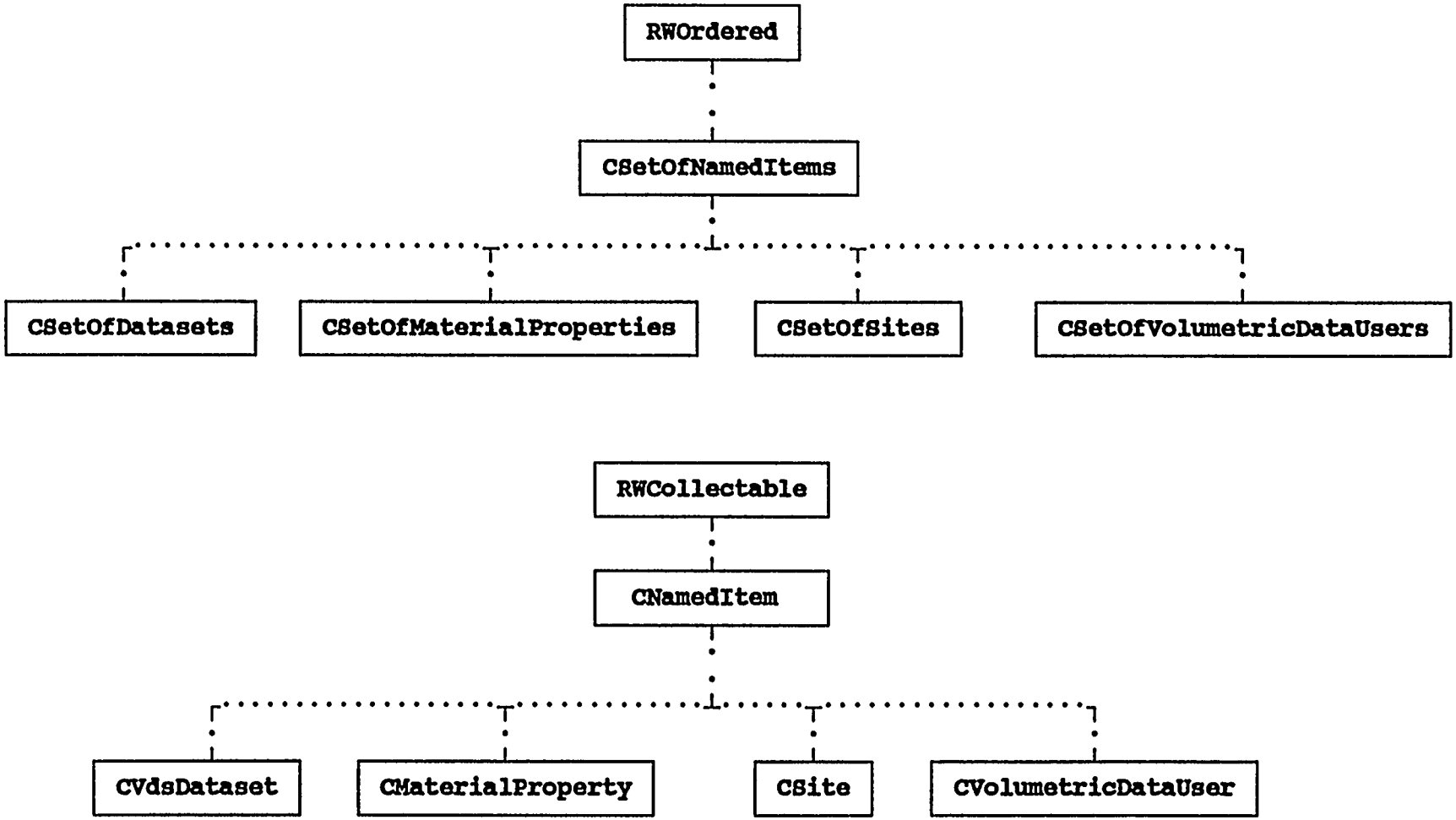


Figure 7-5 VDS Client Interface CSC OOD Diagram #1

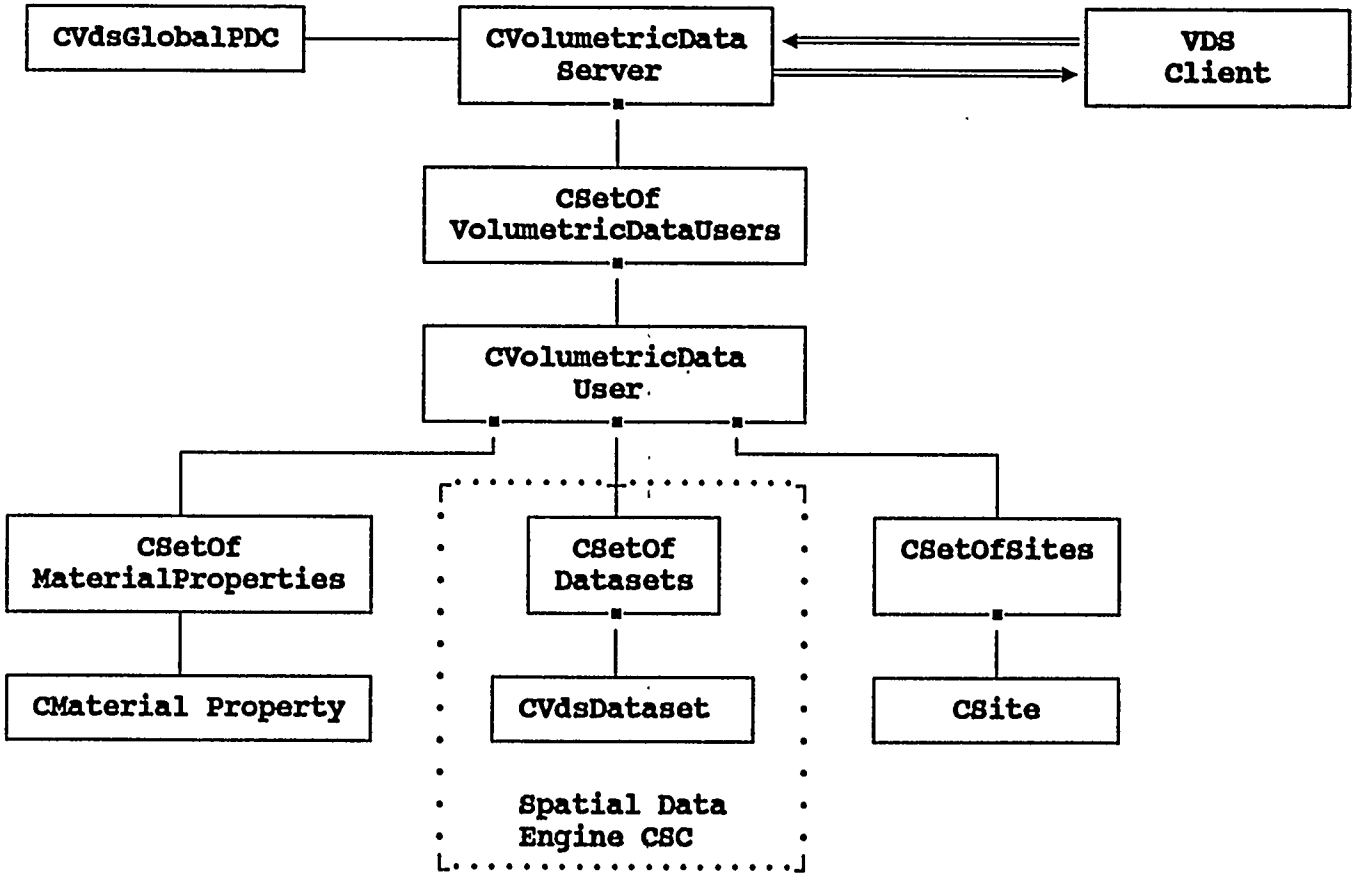


Figure 7-6 VDS Client INterface CSC OOD DIagram #2

CSetOfVdsDatasets: A class that implements a collection of CVdsDataset objects. The class is a part of the Spatial Data Engine CSC (see section 7.4), but is introduced here to illustrate the connection between the VDS Client Interface CSC and the Spatial Data Engine CSC.

CSetOfMaterialProperties: A class that implements a collection of CMaterialProperty objects. This class was previously defined in section 6.3 and is redefined here for completeness.

CSetOfVolumetricDataUsers: A class that implements a collection of CVolumetricDataUser objects.

CSetOfSites: A class that implements a collection of CSite objects. This class was previously defined in section 6.3 and is redefined here for completeness.

CSite: A class that encapsulates all the waste site (workspace or task space) specific data and control parameters. One instance of this class exists for each waste site known to the ICERVS system. The CSite class is derived from CSiteParameters class. This class was previously defined in section 6.3 and is redefined here for completeness.

CVolumetricDataClient: This class encapsulates and hides the implementation details of the VDS client. This class provides all communications with the VDS server. Methods of this class provide an application with complete access to the functions supported by the VDS server.

CVolumetricDataServer: This class encapsulates and hides the implementation details of the VDS Server. This class provides all communications with the server's clients, receives and dispatches client commands, and sends server reply messages to the client. The VDS Server contains exactly one instance of this class.

CVolumetricDataUser: This class encapsulates a user of the VDS Server. It contains the data objects that are unique to each external VDS client. The CVolumetricDataServer object creates a new instance of this class for each client that issues a VDS_ConnectToServer command and destroys the object when the client issues the VDS_DisconnectFromServer command. This class assists the CVolumetricDataServer object with the execution of client commands.

7.3.3.2. HIC Classes

The Volumetric Data Client Interface CSC contains no HIC classes.

7.3.3.3. Globally Available Data

Figure 7-5 introduced PDC class VdsGlobalPDC that contains pointers to global data objects.. The VdsGlobalPDC class was invented to be a convenient repository of data which will be needed by many other PDC and HIC classes in this and other CSCs. Rather than making each of the data items global, it was decided to encapsulate the concept of global data into a single class and object. This class contains

only static data members and static access functions for the data. Since the data members are static, all instances of the class share the same data members. This means that an object wishing to access the application global data need only instantiate a local object of the VdsGlobalPDC class. When completed, the local object is deleted. Thus, access to application global data is simple and requires minimal interaction.

7.3.4. Major Function Descriptions

The following major functions are performed by the Volumetric Data Client Interface function. Each of the functions is implemented as a method of the CVolumetricDataServer class or its base classes.

Start Client Server Interface: This function will be performed in a method of the CVolumetricDataServer class. All necessary steps are performed to initialize the client-server communications link. Other initializations steps to prepare the CVolumetricDataServer object for operation are also performed. Upon completion of this function, the VDS Server is ready to accept a client.

Terminate Client Server Interface: This function shutdowns the client-server communications link. Communications connections to all clients will be dissolved.

Get A Client Command: This function will await the receipt of a client command. The command may come from the client-server communications link or from the VDS Operator Interface (Visualization and Interaction CSC).

Execute A Client Command: Commands received from the client are passed to this function for execution. If the command is from the General Command set, the CVolumetricDataServer object will execute the command. Otherwise, the command is passed to the appropriate CVolumetricDataUser object for execution. Upon completion of the command, a reply message is sent to the client.

Connect A New Client: This function processes the VDS_ConnectToServer command. A new CVolumetricDataUser object is created, initialized, and added to the CSetOfVolumetricDataUsers object maintained by the CVolumetricDataServer object.

Disconnect A Client: This function processes the VDS_DisconnectFromServer command. The appropriate CVolumetricDataUser object is located and destroyed. The client's communication connection is closed.

Activate Operator Interface: This function processes the VDS_EnableOperatorInterface command. A CVdsMainWindow (see section 7.9) is created, initialized and started. The VDS Main Window will be displayed on the client's display screen.

7.4. Spatial Data Engine CSC Detailed Design

The Spatial Data Engine (formerly Octree Engine) CSC provides the primary means for storing, retrieving, and manipulating volumetric data. It incorporates a major upgrade in the capabilities for integrating and analyzing measurement data with the inclusion of Octree Corporation's TrueSolid software. In addition to providing a controlled interface to the TrueSolid software, this CSC will provide enhanced features such as general cutplane tools, data sculpting, and spatial comparisons between measurement data sets. The Spatial Data Engine CSC is also the gateway to the Property Data CSC. The Spatial Data Engine CSC interfaces to the Volumetric Data Client Interface CSC through the CVolumetricDataUser class. Figure 7-7 shows the OOA diagram for the Spatial Data Engine CSC.

7.4.1. Dataset Interface

The central component in the Spatial Data Engine CSC is the dataset class (CVdsDataset). This class ties together all volumetric data functions. There can be multiple dataset objects (one for each site/dataset combination selected by the operator). The material property class, cutplane class, view class, and tree interface class are all instantiated from the CVdsDataset class. The material property class (CMaterialProperty) provides a list of material properties stored for each dataset object. The cutplane class (CCutplane) provides a set of six predefined cutplanes for each dataset object. The view class (CVdsView) describes the views of the dataset that are being rendered in each view window. The tree interface class (CTreeInterface) provides the interface to the TrueSolid function library.

7.4.2. TrueSolid Interface

The interface to TrueSolid is done via the CTreeInterface class. This class provides the only interface between the Spatial Data Engine CSC and the TrueSolid library. All of TrueSolid's features described below are obtained through this interface class.

TrueSolid provides tools necessary for the building, viewing, and modification of 3D volumetric information. 3D data is supplied in one of four ways: a volume array file; a set of 2D slices which are stacked together to build a volume; a set of 3D points used to build a volume a voxel at a time; or as a CSG description of a solid. ICERVS Phase II will be using the third method (a set of 3D points). After 3D data has been input using one of these four methods, stored octrees can be build or modified a voxel at a time, where the 3D coordinate and the level (the size of the voxel) are given along with a property. These stored octrees can be written on disk for rapid retrieval.

TrueSolid also builds objects using a Constructive Solid Geometry (CSG) approach. These objects are a tree structure of the union, intersection, or subtraction of other CSG objects. At the lowest level of the CSG tree are the primitives, which are currently spheres, cones, cylinders, toroids, planar half-spaces and stored octrees. TrueSolid is used to display the object from any point of view and scaling. Geometric objects will be displayed using CSG objects.

TrueSolid also provides for various classification operations which can be performed on stored octrees. Such operations include: thresholding, connectivity, isodensity surface, gradient computation, etc. The VDS and geometric object analysis functions will use these operations.

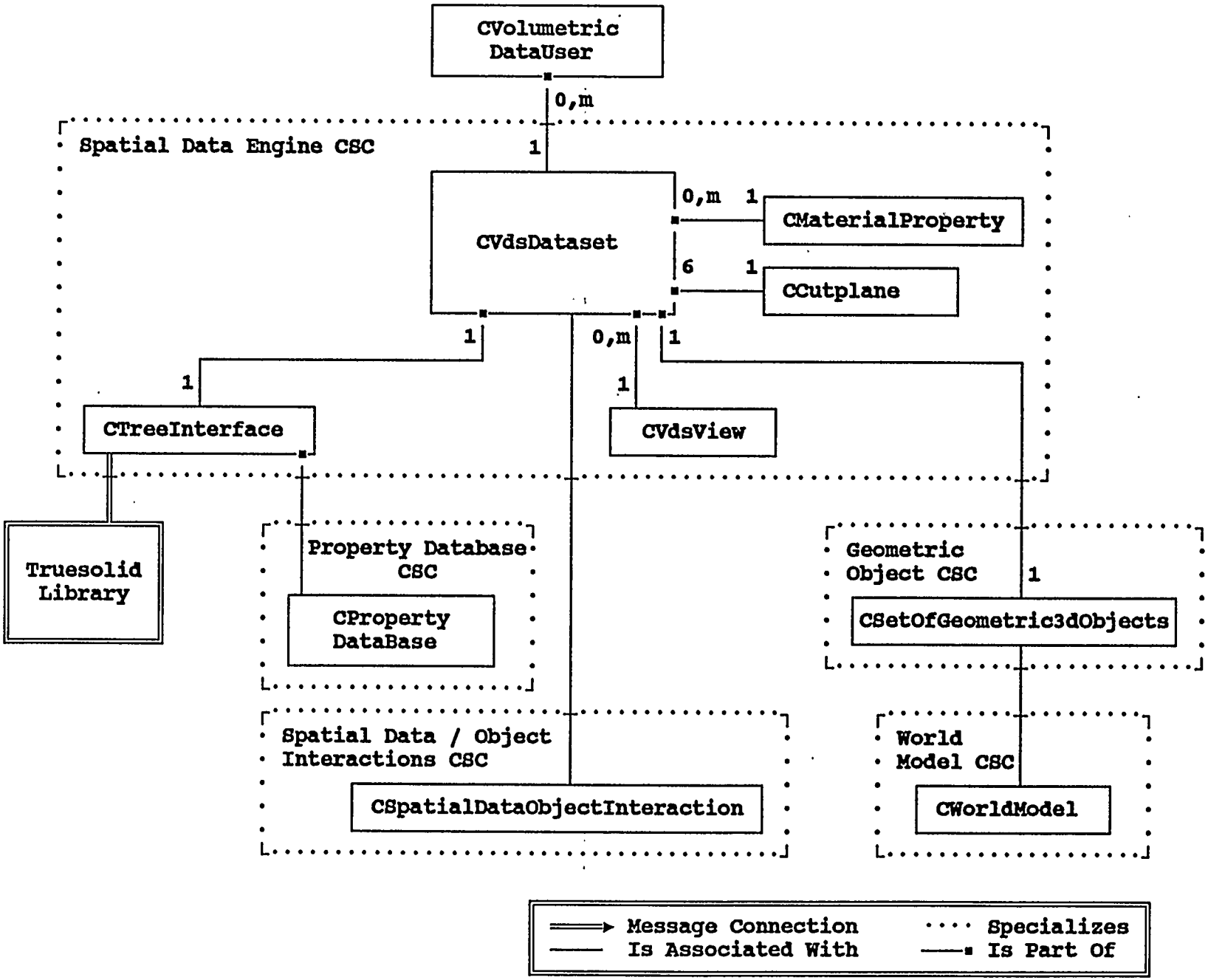


Figure 7-7 VDS Spatial Data Engine OOA Diagram

7.4.3. Class Descriptions

The Spatial Data Engine CSC has only Problem Domain Classes. The Human Interface Classes are contained in the Visualization And Interaction CSC. The Spatial Data Engine CSC OOA diagram of Figure 7-7 has been expanded and specialized to produce the OOD diagrams in Figures 7-8 and 7-9. Figure 7-8 shows the derivation of the basic PDC classes and introduces a set of classes that form collections of the basic classes. (The RWCollectable and RWOrdered classes are from the Rogue Wave Tools^{++} library.) Figure 7-9 combines all the PDC classes and illustrates the Is A Part Of relationships among objects of the various classes. The following describes the Spatial Data Engine CSC classes:

CCutplane: This class encapsulates the concept of a cutting plane which is used to slice the data volume and facilitate the exploration of internal features of that data volume. There are 6 predefined cutplanes which get their default values from *the site parameter file*. Information describing the position and orientation of the cutplane is stored in this class. The values may be changed by the cutplane editor from the VDS main window *Cutplane-Edit* menu item, and/or by the cutplane selection dialog from the VDS view window *Cutplane* menu item.

CMaterialProperty: This class encapsulates the abstract idea of a material property (i.e. a temperature in degrees Fahrenheit). It does not include material property data value (i.e. the value of the temperature). This class was previously defined in section 6.3 and is redefined here for completeness. There is a material property data file for each property in the set. Each property file is named: "<property name>.prp". The prp files are used to build the initial CSetOfMaterialProperties object for each dataset. In the VDS Server, material properties are associated with a dataset.

CSetOfCutplanes: This class implements a collection of CCutplane objects.

CSetOfMaterialProperties: This class implements a collection of CMaterialProperty objects. Instantiated from the CVdsDataset class, it is used to store the set of material properties that exist for a given dataset. New entries are made whenever a new material type is added to the dataset.

CSetOfVdsDatasets: This class implements a collection of CVdsDataset objects. It will be instantiated from a CVolumetricDataUser object and is used to manage the datasets belonging to that user.

CSetOfVdsViews: This class implements a collection of CVdsView objects. It keeps a list of all view windows which are currently open for each dataset. This list of views is needed by the main application when setting up a camera/view connection for snapping a view to the camera angle.

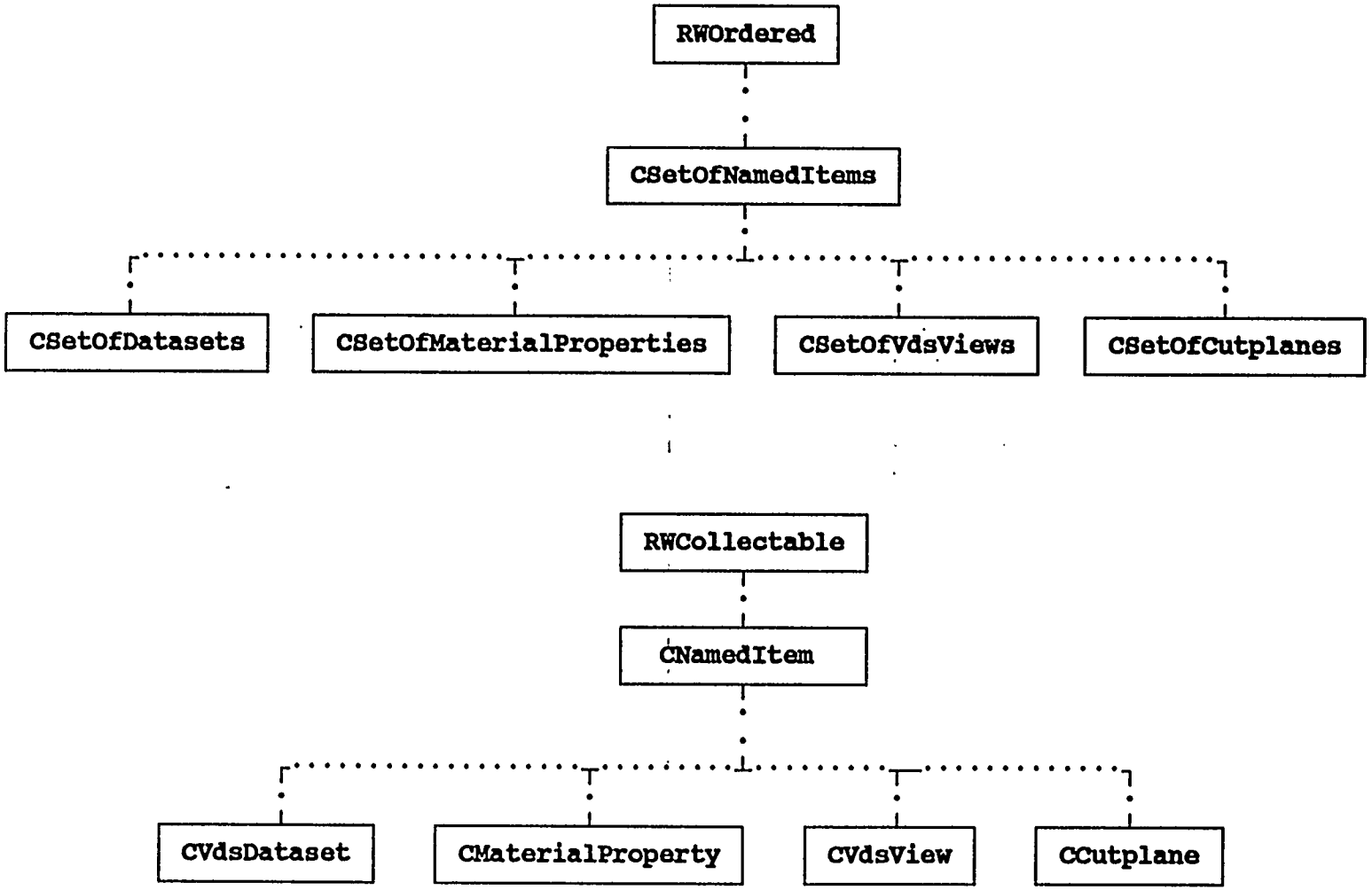


Figure 7-8 Spatial Data Engine CSC OOD Diagram #1

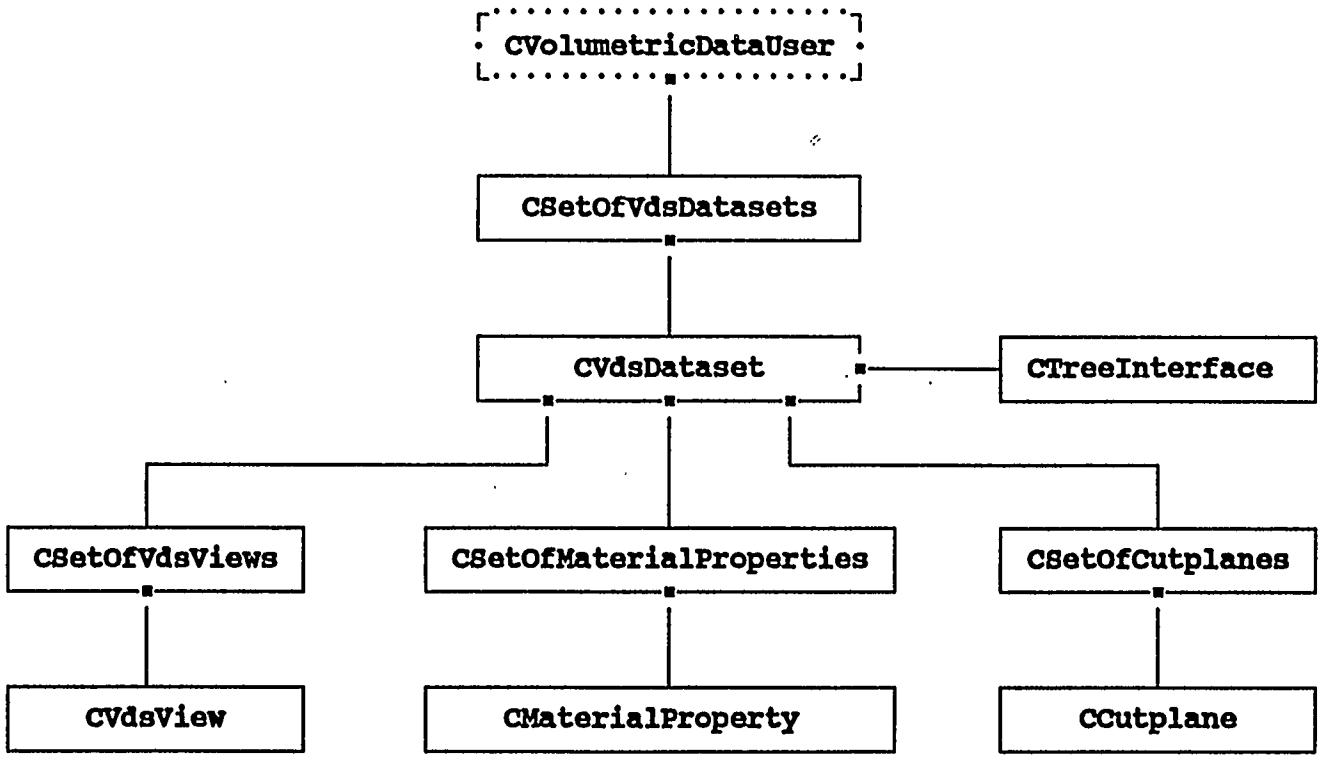


Figure 7-9 Spatial Data Engine CSC OOD Diagram #2

CTreeInterface: This class encapsulates the interface to the TrueSolid library. All octree functions are methods of this class. Below are the major methods in the CTreeInterface class:

SetTreeSize	Sets size of tree in data units:
SetAddTreeLevel	Sets the resolution of the tree (level 1-10)
ConnectView	Connects a view window to an octree
DisconnectView	Disconnects a view window from a tree
DisplayView	Displays a tree in a view window
NewTree	Creates a new tree (one with no points in it)
TreeStat	Shows the statistics of the tree
ReadTree	Reads an octree from disk into memory
SaveTree	Saves an octree in memory to disk
TreeToPostscript	Stores to a postscript file
TreeToASCII	Saves each voxel of an octree in memory to an ASCII file
AddTreePoint	Adds an X,Y,Z,level,node type to an octree in memory
EraseTreePoint	Deletes an X,Y,Z,level from an octree in memory
AddList	Adds a file of points to an octree in memory
AddAndSculptTreePoint	sculpts and adds a set of points to an octree
AddAndSculptList	Sculpts/adds a file of XYZ, level, node type to an octree
SetTreeRegionNodeType	Sets a region in the octree to a given node type
TreeUnion	Unions 2 trees to produce a third temporary tree
TreeIntersect	Intersects 2 trees to produce a third temporary tree
TreeSubtract	Subtracts 2 trees to produce a third temporary tree
TreeTransform	Translate, rotate, or scale a tree to produce a new tree
TurnOnTreeObjects	Turns on all geometric objects stored as CSGs in tree
TurnOffTreeObjects	Turns off all geometric objects stored as CSGs in tree
TreeObjectVolumetricDiff	Performs difference between an object and a tree
TreeObjectConsistencyCheck	Performs consistency check between object and tree

CVdsView: This class encapsulates the information for a specific view of a dataset. Information describing the position and orientation of the view with the data volume is stored within this class. Each CVdsView is associated with a CVdsViewWindow object that displays the dataset.

CVdsDataset: This class encapsulates the concept of a volumetric dataset. A dataset is the main concept in the VDS. It can be thought of as "containing a collection of data that characterizes a waste site at a particular point in time". Within a dataset, the data can be grouped even further by using the notion of data IDs. An ID can be used to separate the dataset into any group so that it can be displayed later with any ID(s) turned on or off in the dataset. This might be used to differentiate between different dig sites or different points in time. The ID is input at the time when data is being entered into a dataset. The material property class, cutplane class, view class, and tree interface class are all instantiated from a CVdsDataset object.

7.4.4. Major Function Descriptions

The Spatial Data Engine CSC major functions can be broken into 5 groups: Material Property functions, Cutplane functions, View functions, Dataset functions, and Tree Interface functions. Each group of functions is detailed below.

7.4.4.1 Material Property Function Descriptions

Add Material Property: Adds a new material property to a specified dataset. A material property data file (name.prp) is created to hold the data for the property. A new CMaterialProperty object is instantiated and added to the CSetOfMaterialProperties object for the dataset.

Delete Material Property: Deletes a material property from a specified dataset. All associated material property data files (.prp) are deleted. The CMaterialProperty object is removed from the CSetOfMaterialProperties object.

Get List of Material Properties for a Dataset: This function is called to obtain a list of material properties for a specific dataset. A call is made to the CSetOfMaterialProperties object to get the list of properties for a given dataset.

Get List Of Material Property Types: This function is called to obtain a list of material property types. The Master Material Properties File is scanned by the CVolumetricDataUser object to generate the list.

Modify Material Property Parameter: Allows for the redefinition of material property parameters such as name, units, conversion factor, etc. A call is made through the CVdsDataset object to a CMaterialProperty object to set the value of one of its parameters.

7.4.4.2 Cutplane Function Descriptions

GetListOfCutplanes: This function is called to obtain a list of cutplanes for a specific dataset. A call is made through the CVdsDataset object to the CSetOfCutplanes object to get the list of cutplanes for a given dataset.

Modify Cutplane Parameter: Allows for the redefinition of cutplane parameters such as name, type, location, orientation, and etc. A call is made through the CVdsDataset object to a CCutplane object to update one of its parameters with a new value.

7.4.4.3 View Function Descriptions

Close View: This function is used to close a view of a dataset. A call is made through the CVdsDataset object to the CSetOfVdsViews object to delete the specified CVdsView object.

Get List of Views: This function is used to obtain the list of views for a specific dataset. A call is made through the CVdsDataset object to the CSetOfVdsViews object to get the view list for a specified dataset.

Get View Display Data: This function is used to obtain the data to display in the associated CVdsViewWindow. A call is made through the CVdsDataset object to the CTreeInterface object to scan the volumetric/property data files and to return the displayable data for the specified CVdsView.

Modify View Parameter: This function allows the redefinition of view parameters such as name, description, transformation factors, etc. A call is made through the CVdsDataset object to a CVdsView object to set the value of one of its parameters.

Open View: This function is initiated to create a new view on a dataset. A call is made through the CVdsDataset object to the CSetOfVdsViews object to add a new CVdsView object for a specific dataset. Default parameters for the view are read from disk.

7.4.4.4 Dataset Function Descriptions

Add Data Points To Dataset: This function is used to add data to a dataset. Single points, lists of points, or external files of data points may be added to a dataset. A call is made through the CVdsDataset object to the CTreeInterface object to perform the actual addition of data to the appropriate octree(s) associated with the dataset.

Close A Dataset: This function is used to close a dataset and remove it from the list of open datasets. The Save A Dataset function should be used, if necessary, prior to closing the dataset. All CVdsView object, if any, for the dataset will be closed and then deleted. The CVdsDataset object is removed from the CSetOfVdsDatsets and deleted..

Define a New Dataset: This function is used to create a new dataset for a site. A new dataset object (CVdsDataset) is instantiated which sets up a dataset directory and all of the necessary files. The CVdsDataset object is added to the CSetOfVdsDatasets.

Delete A Dataset: This function is used to delete open dataset. The data is deleted by removing the dataset directory and all of its files. The corresponding CVdsDataset object is deleted from the CSetOfVdsDatasets object.

Get List Of Datasets: This function is used to obtain the list of datasets for a site. A call is made through the CVolumetricDataUser object to the CSetOfVdsDatasets object to get the list of datasets.

Modify Dataset Parameter: This function allows the redefinition of dataset parameters such as name, description, etc. A call is made to a CVdsDataset object to set the value of one of its parameters.

Open An Existing Dataset: This function is used to open and read a dataset into memory. A new dataset object (CVdsDataset) is instantiated which reads in the associated data files. The CVdsDataset object is added to the CSetOfVdsDatasets.

Save A Dataset: This function is used to write the data associated with a dataset to the disk for permanent storage. The data is saved by calling the CTreeInterface method SaveTree.

7.4.4.5 Tree Interface Function Descriptions

Initialize Octree: This function is used to initialize a new tree in memory.

Write Octree Data To Disk File: This function is used to write an octree from memory to a disk file. A call is made to CTreeInterface::SaveTree to transfer the data from the tree structure in memory to the disk file. The octree data is written to disk in binary format matching the internal format of TrueSolid. In addition, any property data that has changed will also be written to disk in a file named with the property type as its filename (i.e. Temperature.prp).

Read Octree Data From Disk File: This function is used to read an octree from disk into memory. A call is made to CTreeInterface::ReadTree to transfer the data from disk to the tree structure in memory. The appropriate type of property will be read along with the dimensional data.

Add Single Point To Octree: This function is used to add a single point to an octree. The point includes x, y, z, tree level, node type, and property value. A call to CTreeInterface::AddPoint is made to insert the new point into the tree. The data for the point must be in ICERVS internal units.

Erase Single Point From Octree: This function is used to erase a single point from an octree. The point to be erased is defined as x, y, z and tree level. A call to CTreeInterface::ErasePoint is made to delete the point from the tree. The data for the point must be in ICERVS internal units.

Add List of Points To Octree: This function is used to add a list of points to an octree. This list is actaully a set of single points which will be added with theCTreeInterface::AddPoint method. A call to CTreeInterface::AddList is made to handle the reading of the list and the insertion of the list of points into the tree. The data for the points must be in ICERVS internal units.

Add With Sculpting Single Point To Octree: This function is similar to the Add function. A call to CTreeInterface::AddAndSculptPoint is made to insert the new point into the tree. When added, all points above the new point will be cleared. This function is used specifically when adding points from sensor data. As surface points are added, it is assumed that the space between the sensor and the surface is empty. Therefore, the state of nodes in the octree which represent that space are automatically set to empty as the new data points are added to the tree. The data for the point must be in ICERVS internal units.

Add With Sculpting List Of Points To Octree: This function is similar to the AddList function. A call to CTreeInterface::AddAndSculptList is made to insert a file of points into the tree. Note that the data contained in the file must be in ICERVS internal units. The Add With Sculpting Single Point function will be used to add the points one at a time. When added, all points above the new point will be cleared.

Set RegionNodeType: This function is used to set a region of points in the octree to a given node type (i.e. empty, partial, full). A call to CTreeInterface::SetRegionNodeType is made to change the node type for the specified region in the tree.

Perform Tree Scan To Print Tree Statistics: This function is used to print the octree statistics. A call to CTreeInterface::TreeStatistics is made to scan the tree and print the statistics to an intermediate disk file. Upon return from the print routine, the intermediate file will be passed to a CBrowseWindow object for viewing.

Perform Tree Scan To Display Tree In View Window: This function is used to display an octree in a given view window for on of the following reasons:

1. Translation of any view
2. Rescaling of any view
3. Movement of the cutplanes in any view
4. Redefinition of view parameters (color, level, etc.)
5. Addition of new data in any view

A call is made to CTreeInterface::DisplayTree to extract view-specific parameters (window size, translation offsets, scaling factors, display level, cutplane locations, etc.) and pass these parameters to the associated octree using a variety of CTreeInterface functions. In addition, the CTreeInterface::DisplayTree routine will pass the view shared parameters (node colors, etc.) to

CTreeInterface. Finally, CTreeInterface::DisplayTree is called to generate an Xlib memory buffer of visible octree points. The view parameters passed by CTreeInterface are used to determine the visibility of each tree point. The data for a point (coordinates, property, state, etc.) will be displayed in the operator's external units.

Perform Tree Analyze Function: This function is used to perform octree analysis functions. A call is made to one of the following methods: CTreeInterfac::TreeUnion, CTreeInterface::TreeIntersection, CTreeInterface::TreeSubtraction to perform a function on two trees to produce a third temporary tree for the purposes of display.

Perform Tree Transform Function: This function is used to transform an octree by either scaling, translation, or rotation). A call is made to the method CTreeInterface::Transform to perform a transform function on the tree to produce a new tree. for the purposes of display.

Create a Postscript File From Octree: This function is used to generate a postscript file from an octree being display in a view. A call to the method CTreeInterface::TreeToPostscript is made to generate the postscript file.

Create an ASCII File From Octree: This function is used to generate an ASCII file from an octree being displayed in a view. A call to the method CTreeInterface::TreeToASCII is made to generate the ASCII file.

7.5. Geometric Object CSC Detailed Design

The Geometric Object (formerly Object Modeling) CSC provides for the creation, modification, and storage of geometric objects. It includes upgrades to the tools for creating geometric objects, including more general prismatic shapes and geometric primitives such as rectangular parallelepipeds, cylinders, and spheres. Two sets of geometric objects are maintained by the system, including:

1.	Master Object Templates File (object.mlb)	System-wide default object templates.
2.	Geometric Object File (object.dic)	Dataset specific set of user defined objects.

Most of the software for this CSC was implemented under Phase I. Most software class names have been changed to avoid the ambiguity of the term "model object". The term "geometric object" is now being used. Additional geometric object types are being added and the 2D geometric classes have been incorporated into this CSC.

7.5.1. Geometric Object Representation

The storage of a list of vertices for a front and rear face is the approach taken to represent a geometric object. These faces will subsequently represent the plane in which they lie thus allowing for non-parallel faces, skewed and/or tapered objects, and many other non-regular shapes not present in Phase I. Once

created, the object will utilize geometric transforms to scale, rotate, or translate itself to a desired position within a coordinate system.

7.5.2. Object Display and Editing Interfaces

After creation, an object will be displayed with the assistance of the TrueSolid package. If the object option for wireframe is selected, the CSG tree will be built as the union of line segments connecting the vertices. If a solid object is requested then the CSG tree will be the intersection of the planes of each side of the prismoid shape including the two faces. TrueSolid supports the construction of solid objects such as cones, cylinders, and spheres. A wireframe picture of these types of objects will be a prismoid shape with a predetermined number of vertices per face to simulate a circular object. The approach used will be to build a CSG tree out of the object's representation, convert the CSG to an image, and then pass it along as an image to a view window which may be display other the site data. Refer to Figure 7-10 for an illustration of some objects displayed using TrueSolid.

For editing, the view window will by placed under the control of the RWCanvas package to allow tracking of individual items. Once in the view window, the objects can be selected and edited. Editing functions can involve translating and rotating the object, or the reshaping, rotating, or resizing of an object's face.

7.5.3. Class Descriptions

The Geometric Object CSC has only Problem Domain Classes. The Human Interface Classes are contained in the Visualization And Interaction CSC. The Volumetric Data Subsystem OOA diagrams of Figures 7-2 and 7-3 have been expanded and specialized to produce the Geometric Object CSC OOD diagrams in Figures 7-11 through 7-13. Figures 7-11 shows the derivation of the basic 2D and 3D PDC classes. Figure 7-12 introduces a set of classes that form collections of the basic classes.

The RWCollectable and RWOrdered classes are from the Rogue Wave Tools^{++} library. Figure 7-13 combines all the PDC classes and illustrates the Is A Part Of relationships among objects of the various classes.

CSetOf3DGeometricObjects: A collection class that contains all the 3D geometric objects for a particular dataset for a particular site. This class encapsulates all accesses to its set of geometric objects. This class is also used to hold the system-wide Master Object Templates. It is derived from CSetOfNamedItems, a general purpose collection class designed and used throughout the ICERVS program.

CGeometricObjectParameters: The abstract base class for all 2D and 3D geometric object classes. This class provides all the common behavior for geometric objects, including a common type that can be handled by the collection classes. The class data members include object name, type, notes, creation date/time, operator identification, object category name, color of the object, etc.. All other geometric object classes must be derived from CGeometricObjectParameters. This class is itself derived from CNamedItems.

Figure 7-10 Geometric Object Editing and Display

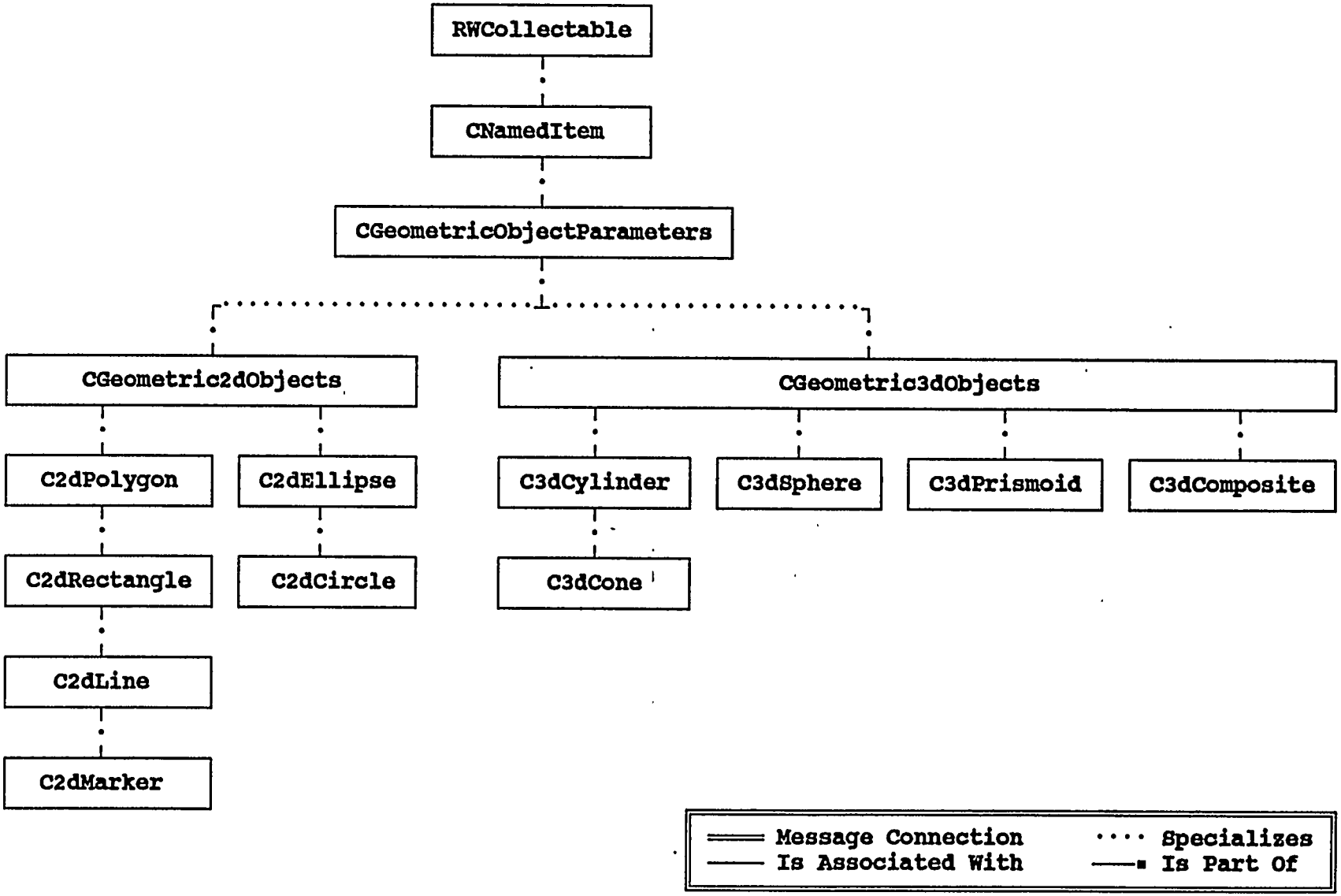


Figure 7-11 Geometric Object CSC OOD Diagram #1

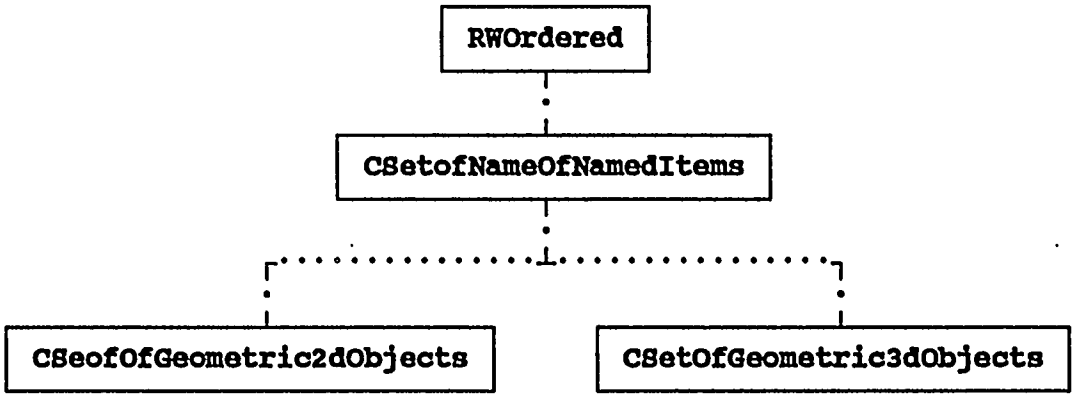


Figure 7-12 Geometric Object CSC OOD Diagram #2

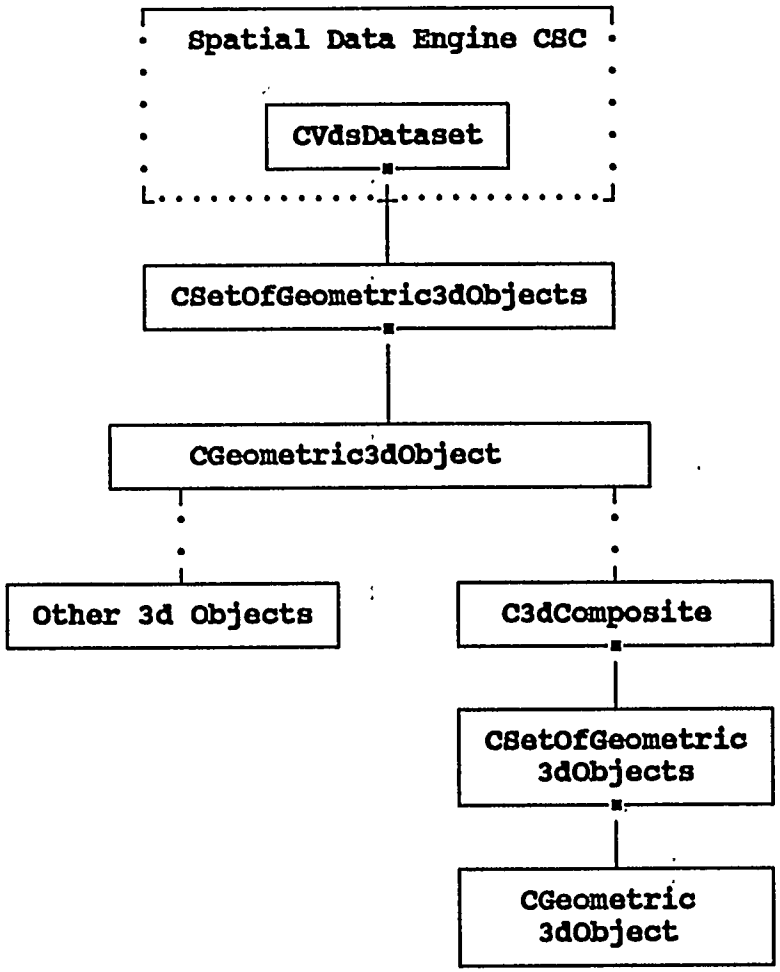


Figure 7-13 Geometric Object CSC OOD Diagram #3

Figure 7-14 VDS Geometric Object OOD Diagram #4

BLANK PAGE

CGeometric3dObject: The abstract base class for all 3D geometric object classes. This class provides all the common behavior for 3D geometric objects, including a common type that can be handled by the 3D collection classes. Other 3D geometric object classes specialize this class by providing additional data to define the specific geometry of the object. This class is derived from CGeometricObjectParameters. The extra data members of this class include the list of coordinates for the front and rear face vertices.

C3dPrismoid: A class that implements a type of CGeometric3dObject characterized by two polygonal faces. There are no extra data members for this class; it is represented solely by the list of vertices inherited from the CGeometric3dObject class.

C3dCylinder: A class that implements a regular 3D geometric cylinder. The extra data members of this class include the center of the base, the diameter of the base and the height of the cylinder.

C3dCone: A class that implements a regular 3D geometric cone. The extra data members of this class include the center of the base, the diameter of the base and the height of the cone. It is derived from the C3dCylinder class and sets all the rear vertices to the center of the face.

C3dSphere: A class that implements a 3D geometric sphere. The extra data members of this class include the center of the sphere and its diameter.

C3dPlane: A class that implements a planar geometric object. The extra data members of this class include a point on the plane and the normal vector to the plane.

C3dComposite: A class that implements a composite 3D geometric object that is composed of several other 3D geometric objects. The extra data members of this class include a list of objects that comprise the group.

CSetOf2DGeometricObjects: A collection class that contains a number of 2D geometric objects and encapsulates all accesses to its set of geometric objects. This class is used mainly in support of orthogonal view by the Visualization And Interaction CSC.

CGeometric2dObject: The abstract base class for all 2D geometric object classes. This class provides all the common behavior for 2D geometric objects, including a common type that can be handled by the 2D collection classes. Other 2D geometric object classes specialize this class by providing additional data to define the specific geometry of the object. 2D geometric objects are used internally within the Visualization And Interaction CSC to support orthogonal view windows. This class is derived from CGeometricObjectParameters.

C2dPolygon: A class that implements a polygon. It is derived from CGeometric2dObject and the extra data members of this class include a list of vertices.

C2dRectangle: A class that implements a rectangle. It is derived from C2dPolygon and forces the number of vertices to four.

C2dLine: A class that implements a line. It is derived from C2dPolygon and forces the number of vertices to two.

C2dMarker: A class that implements a 2d marker. It is derived from C2dPolygon and forces the number of vertices to one.

C2dCircle: A class that implements a circle. It is derived from CGeometric2dObject and the extra data members of this class include a center point and a radius.

C2dEllipse: A class that implements an ellipse. It is derived from CGemometric2dObject and the extra data members include a center point, two radii, and an elevation angle.

C2dComposite: A class that implements a composite geometric 2D object that is composed of several other 2D geometric objects. The extra data members of this class include a list of other objects that comprise the group.

7.5.4. Major Function Descriptions

The functions of the Geometric Object CSC are used internally by the ICERVS software subsystem. No operator will directly interface with these functions. Instead, the operator will interact with some user interface and that user interface will interact with the Geometric Object CSC by issuing commands to the VDS Server. Refer to Table 7-11 for a list of VDS geometric object commands. The Geometric Object CSC major functions are described below:

7.5.4.1. Read Geometric Objects List/Template Files

This function is internal to the CSetOfGeometric3dObjects class. When an instance of either class is created, the constructor for the object will read the appropriate disk file and load the geometric object data.

7.5.4.2. Write Geometric Objects List/Library Files

This function is internal to the CSetOfGeometric3dObjects class. When an instance of either class is destroyed, the destructor for the object will write the appropriate disk file and store the set of objects to disk. If no changes have been made to the object data, the file write process will be bypassed. The representation of the objects will be written to disk in the same manner as that of the site data where an instance of the CConfigMgr class will generate an ASCII file containing all the necessary information to regenerate the objects.

7.5.4.3. Add New 3D Object To List or Library

Addition of new geometric objects to the list or library involves only the creation of a new instance of CGeometric3dObject (actually one of its derived classes) and the addition of the new instance to the list or

library. This function of performed in response to a VDS_CreateObject command. A geometric object with default parameter values is created and placed into the appropriate set of objects. The SetObjectParameter command must be used to define the new object's actual parameter values. An objectId is returned by this function.

7.5.4.4. Delete 3D Object From List or Library

Deletion of a geometric object from the list or library involves only the removal of the object from the list or library. This function is performed in response to a VDS_DeleteObject command.

7.5.4.5. Modify 3D Object In List or Library

Modification of a geometric object in the list or library involves the extraction of the objects data, modification of that data, and re-insertion of the object into the list or library. The VDS_GetObjectParameter and VDS_SetObjectParameter commands are used to perform this function.

7.5.4.6. Associate (add or modify) Text With Geometric Object

This function allows for the association of text with the 3D geometric objects that currently exist in the geometric object list. Aside from the object name, description, the creator name, category name, and the date and time the object was first created, the ICERVS system allows a variable length text block to be associated with each object. Any information desired may be entered into the text block. The VDS_SetObjectParameter command is used to defined the text block.

7.5.4.7. Output Text Report Of Geometric Objects

This function allows for the browsing and optional printing of any or all 3D polyhedral objects that currently exist in the geometric object list/library. The output consists of an object name and description, the operator name who created the object, the date and time the object was first created, the category of the object, the color of the object, and a list of vertices that describe the object. The output is formatted and written to an ASCII disk file.

7.6 Spatial Data/Object Interaction CSC Detailed Design

The Spatial Data/Object Interaction CSC provides a link between the Spatial Data Engine CSC and the Geometric Object CSC. It incorporates capabilities to enable mathematical operations between octree data and geometric object data. This CSC will allow an operator to, for example, compute the volume of octree material located inside a selected geometric object and automatically scan the octree space to identify any geometric objects that are void (contain no octree material). Refer to the OOA diagram of Figure 7-7 for an illustration of this CSC and its relationship to the other parts of the Volumetric Data Subsystem.

7.6.1 Dataset Interface

The CVdsDataset class coordinates the link between the Spatial Data Engine CSC and the Geometric Object CSC. The CVdsDataset object receives the commands from the client/server connection and instantiates the CSpatialDataObjectInteraction class to perform the function. The CSpatialDataObjectInteraction object gets all the necessary octree information from the CTreeInterface

object in the Spatial Data Engine CSC and all the necessary geometric object information from the CGeometric3dObject objects in the Geometric Object CSC in order to perform the requested function.

7.6.2 Class Descriptions

There is one class in the Spatial Data/Object Interaction CSC (CSpatialDataObjectInteraction). This class contains methods to perform all the ICERVS Phase II requirements for the interaction between the volumetric data and geometric objects.

CSpatialDataObjectInteraction: Encapsulates all Spatial Data/Object Interaction functions as methods within the class. The following methods are contained in this class:

ComputeObjectVolumetricDifference: Computes the volume of octree material located inside a specified geometric object. Returns the results of the computation.

PeformObjectConsistencyCheck: Scans the octree space to identify any geometric objects that are void (contain no octree material). Returns a list of inconsistent objects.

ConvertGeometricObjectForDisplay: Converts a CGeometric3dObject into a TrueSolid CSG. Geometric Objects may converted into a wire frame or solid model format. Refer to paragraph 7.5.2 for additional details relating to the display of geometric objects.

7.6.3 Major Function Descriptions

Compute Volumetric Difference Between Octree and Object: The ObjectVolumetricDifference command is sent to the VDS Server and passed on to the CVdsDataset object which instantiates a CSpatialDataObjectInteraction object to execute the command. The CSpatialDataObjectInteraction object accesses the CVdsDataset's CTreeInterface and the CSetOfGeometric3dObjects to obtain volumetric and geometric object information respectively.

The octree is known to Truesolid as a memprim, and the geometric object is known as a CSG. The volume of both these things can be computed by setting the region of each of them to a different node type using Truesolid's *ts_setregion* library function. Then all the nodes with the specified node types just set can be retrieved using Truesolid's *ts_forallnodes* library function with a specified node type. The volume of all the retieved nodes for the tree can be added up and displayed in a browse window. Similarly, the volume of the geometric object can be obtained and displayed. Finally, the difference of the two volumes can be computed and displayed.

The output from this function will be displayed in a browse window containing the volumes of the octree and the objects; and the volumetric difference between the two. In addition, the nodes contained in both volumes will be colored a different color to show the volumetric difference

graphically. When the browse window is exited, the color of the tree and object will return to the state previous to executing this function.

Perform consistency Check Between Octree and Object: The ObjectConsistencyCheck command is sent to the VDS Server and passed on to the CVdsDataset object which instantiates a CSpatialDataObjectInteraction object to execute the command. The CSpatialDataObjectInteraction object accesses the CVdsDataset's CTreeInterface and the CSetOfGeometric3dObjects to obtain volumetric and geometric object information respectively.

The octree is known to Truesolid as a memprim, and the geometric object is known as a CSG. A consistency check of a given object can be computed by setting the region of the CSG object to an empty node type (type=0) using Truesolid's *ts_setregion* library function. Then, if no empty nodes are found in the tree (using Truesolid's *ts_forallnodes* library function with node type zero specified), it is assumed that the object is floating in space.

The output from this function will be displayed in a browse window containing the result of the consistency check bewteen the octree and the specified geometeric object (i.e. floating in space or not floating in space). In addition, if the geometric object is found to be floating, it will be colored in a bright contrasting color to show graphically where the problem object is located. The operator can opt to permanently make the object this color to be reminded that the object must eventually be deleted from the system. When the browse window is exited, the color of the floating object(s) will return to the state previous to executing this function, unless the operator opted to color them permanently.

Convert Geometric Object For Display: Object rendering is performed by converting the geometric object information into TrueSolid CSG trees. Both wire frame and solid modeling formats are provided. Refer to paragraph 7.5.2 for additional details.

7.7. World Model Data Interface CSC Detailed Design

The World Model Data Interface CSC will convert ICERVS geometric modeling objects created by the Geometric Object CSC into a specified form such as IGES, STEP, or IGRIP in order to provide geometric world models to robotic controllers. Only one format (IGRIP) will be supported in ICERVS Phase II. Future capabilities will support input of geometric object information from architectural systems and possible integration with a robotic controller world model. Refer to the OOA diagram of Figure 7-7 for an illustration of this CSC and its relationship to the other parts of the Volumetric Data Subsystem.

7.7.1. Geometric Object Database

The World Model Data Interface will be implemented as a single class which contains methods to perform the conversions. The World Model Data Interface CSC accesses the Geometric Object Database which

stores all the object data and converts the objects to a format compatible with another object storage system. The Geometric Object Database consists of a geometric object dictionary file (object.dic) and a geometric object library file (object.lib) for each dataset. The object dictionary file contains all the information about the defined objects for a particular dataset. The object library file contains predefined shapes which can be used for generating objects. Access to the Geometric Object Database is encapsulated by the CSetOfGeometric3dObjects class, which contains methods to read, write, list, and view the objects.

7.7.2. Class Descriptions

There is one class in the World Model Data Interface CSC (CWorldModel). This class contains the methods to perform all the phase II requirements for the world model.

CWorldModel: This class is the main class for the world model CSC. It contains methods to perform the various conversions. It includes information that is common to all the methods listed below.

ExportToIGRIP: This export class encapsulates the conversion of geometric objects stored in the ICERVS geometric object file to a file in the IGRIP file format.

ExportToSTEP: This export class encapsulates the conversion of geometric objects stored in the ICERVS geometric object file to a file in the STEP file format. This method will not be implemented in Phase II;

ExportToIGES: This export class encapsulates the conversion of geometric objects stored in the ICERVS geometric object file to a file in the IGES file format. This method will not be implemented in Phase II.

7.7.3. Major Function Descriptions

Convert Geometric Objects to IGRIP Format: This function converts the geometric objects to a file in IGRIP format. A command is sent to the VDS Server and passed on to the CVdsDataset object which instantiates a CWorldModel object. The CWorlModel object interfaces with the CSetOfGeometric3dObjects object to obtain geometric object information. The CWorldModel::ExportToIGRIP() method is called to convert the geometric objects into IGRIP format and to generate an output file. readable by IGRIP.

NEED DETAILS AS TO HOW TO DO THIS. WHAT IS THE FORMAT?

7.8. Property Database CSC Detailed Design

The Property Database CSC has been separated from the Spatial Data Engine CSC to promote system flexibility and modularity. During Phase II, it will remain closely coupled to the Spatial Data Engine

CSC. To accommodate the large amounts and multitudinous varieties of property data in an efficient manner, it is anticipated that a commercial DBMS product will be integrated into the Property Database CSC in the Phase III ICERVS. This CSC and the DBMS will provide for the physical storage/retrieval of property data, while the Spatial Data Engine will be used to provide the spatial aspects of the property data. Refer to the OOA diagram of Figure 7-7 for an illustration of this CSC and its relationship to the other parts of the Volumetric Data Subsystem.

7.8.1. Property Database Structure

For ICERVS Phase II, the Property Database will consist of a set of property files within a site/dataset directory. A separate property file exists for each property type contained in the dataset. For example, if there is property data stored for density and radiation level, then there will be two property files within the dataset directory in which the spatial data is also stored. The property files are named with a filename taken from the material property file (the property type) and the file extension of "prp" (i.e. density.prp).

When a CVdsDataset object is created, a CSetOfMaterialProperties object is initialized. The CVdsDataset class constructor examines all the files in the dataset directory with the extension of "prp". A CMaterialProperty object is created for each property data file found and inserted into the CSetOfMaterialProperties object. New property types can be added to the dataset during spatial data input functions. Since TrueSolid can handle only one property file at a time, namely density , the property file for the selected property type must be renamed to (<filename>.Density) in order to be read into TrueSolid along with the spatial data.

The Master Material Property File is in CConfig format with one block for each material property type. Each block contains the property name (which will be used for the filename of the property file), property description, unit label, and conversion factors to convert input property values into TrueSolid's internal format.

7.8.2. Class Descriptions

Only one class exists for this CSC. This class encapsulates all functionality of the property database in the Property Database CSC. The following describes the main class in the Property Database CSC::

CPropertyDatabase: This class encapsulates all functionality of the property database in the Property Database CSC. This isolates the property data functions from the rest of the system so that Phase III can easily incorporate a commercial property database into the system.

7.8.3. Major Function Descriptions

Read a Property Type: This function is used to read a property from the property database into memory along with the octree (dimensional) data. The default view display parameter for property type is used to determine the type of property to be read in with the dimensional data.

This default value is usually "Dimensional", meaning that no property type will be read. However, this can be overridden by the operator during view creation time. This function is actually part of the CTreeInterface class (CTreeInterface::ReadTree) which transfers the data from disk to the tree structure in memory. A command is sent via the client to the server and passed on to the CVdsDataset class where the function is executed using the CTreeInterface class in the Spatial Data Engine CSC. The CTreeInterface class calls a method in the CPropertyDatabase class which is part of the Property Database CSC. The CPropertyDatabase class is in charge of renaming the appropriate property file to be compatible with TrueSolid as follows:

copy radiation.prp <filename>.Densities.
where filename is the name of all octree files within the dataset directory

Only one property can be read in per octree. Once the appropriate property file is renamed, it is read in along with the octree data using Truesolid's *ts_load_octree* library function. Then a threshold is applied to the octree which effectively changes each node's type based on a given min/max range. This is done using Truesolid's *ts_threshold library* function. When the predefined rainbow threshhold file is now applied to the octree, each node will be colored based on property value.

Write a Property Type: This function is used to write a property from memory to the property database. This function is actually part of the CTreeInterface class (CTreeInterface::SaveTree) which transfers the data from the tree structure in memory to disk. A command is sent via the client to the server and passed on to the CVdsDataset class where the function is executed using the CTreeInterface class in the Spatial Data Engine CSC. The CTreeInterface class calls a method in the CPropertyDataBase class which is part of the Property Database CSC. The CPropertyDatabase class is in charge of renaming the appropriate property file from TrueSolid naming convention to ICERVS property naming convention. to be compatible with TrueSolid as follows:

copy <filename>.Densities. radiation.prp
where filename is the name of all octree files within the dataset directory

Convert a Property Type: This function is used to convert a property value from a raw sensor reading to a number in Truesolid internal format. This conversion is done at the point in time when the property data is being added to the octree. A command is sent via the client to the server and passed on to the CVdsDataset class where the function is executed using the CPropertyDatabase class in the Property Database CSC. The following steps must be performed:

- Apply the units conversion stored in the material property file to the raw data

- Convert the property value to octree internal format by using the sensor's min/max range and a property internal range (-32767 to 32767)

7.9 Visualization And Interaction CSC Detailed Design

The Visualization and Interaction CSC implements a standard user interface for the Volumetric Data CSCI, which supports all the common functions for the CSCI. This CSC will produce one or more Motif windows on the client application's display screen from which the operator may interact with and manipulate the application's volumetric, property, and geometric object databases. Figure 7-15 contains an OOA diagram of this CSC.

7.9.1 Operator Interface Description

Almost all of the of code for this CSCI will be for the windows, menus, buttons, and other GUI elements of the operator interface. The Uniras UIMX GUI builder tool will used to layout the screen and generate most of the code. The callback functions for the menu buttons will typically issue one or more VDS Server commands and display the command results in a appropriate GUI window. The following paragraphs describe the major aspects of the operator interface.

7.9.1.1 Volumetric Data Window

The Volumetric Data Window contains Volumetric Data Subsystem's main menu. This window is activated in response to the VDS_EnableUserInterface command. The menu and function buttons on this window implement all the functionality for interfacing to the volumetric database (viewing the data, inputting the data, analyzing the data, cutplaning the data, and reading/writing datasets). Most functions in this window act upon a current dataset. The current dataset is selected from the Volumetric Data Window's pop-down list of datasets. Figure 7-16a contains an example of the Volumetric Data Window.

7.9.1.2 View Windows

The VDS View Window contains the functionality for viewing data volumetric data and for interacting with that data. A VDS View Window is activated from the VIEW menu on the Volumetric Data Window. A view window displays dimensional and/or property data for a selected dataset. Multiple view windows may be created for a dataset Each view window depicts the data from a different view point (i.e. orthogonal or arbitrary view). The view window can be manipulated (scaled and translated) with the sliders located on the two sides and bottom of the view window The data within the view window can be rotated (arbitrary view only) by clicking on the "R" button on the menubar and bringing up a rotation rectangle which allows the operator to rotate the view point via the middle mouse button. The view window can be cutplaned or sectioned by clicking on the "C" button to activate the cutplane manipulation controls. The view window provides six independent cutplanes. In addition, the view window can contain geometric objects.

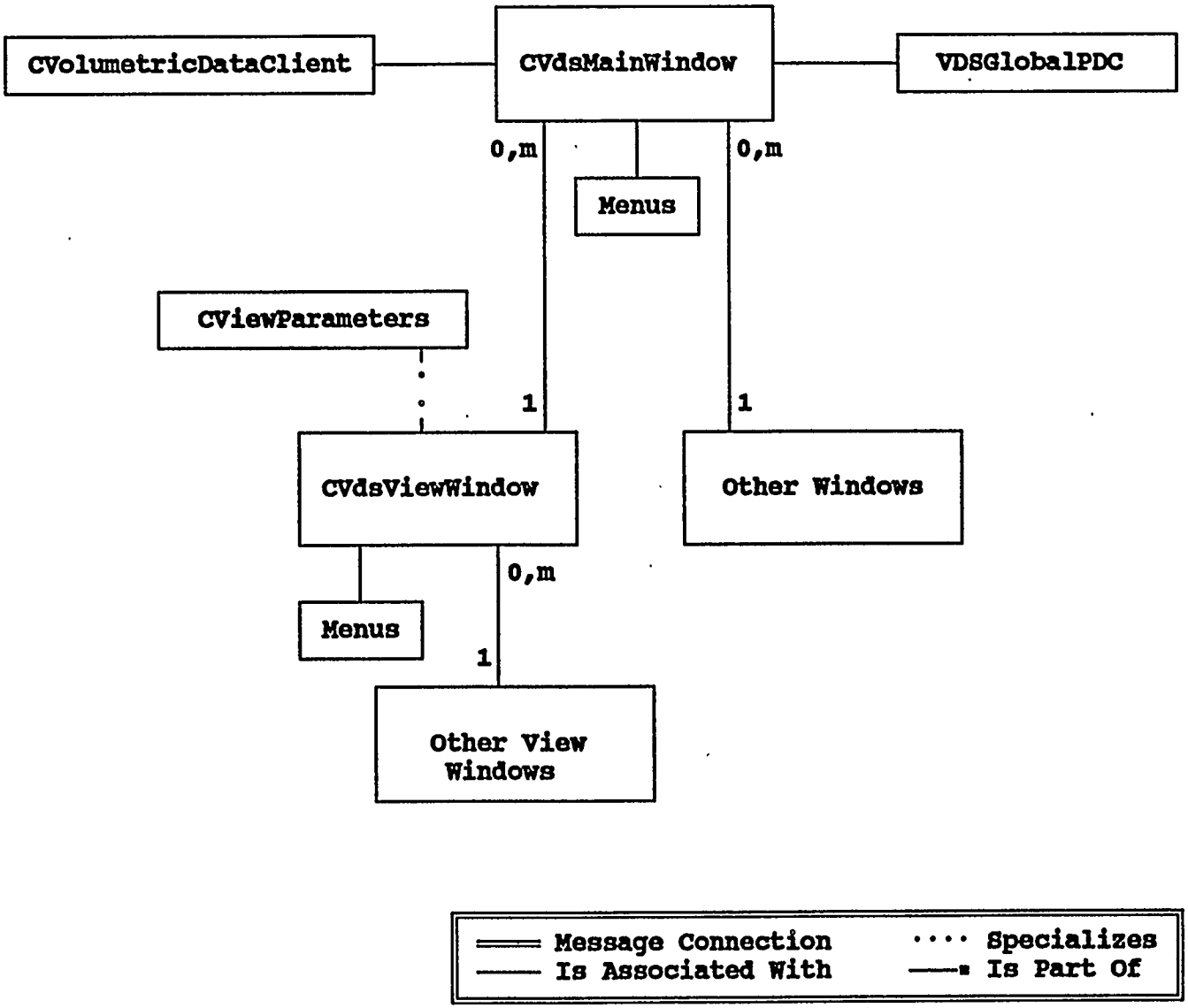
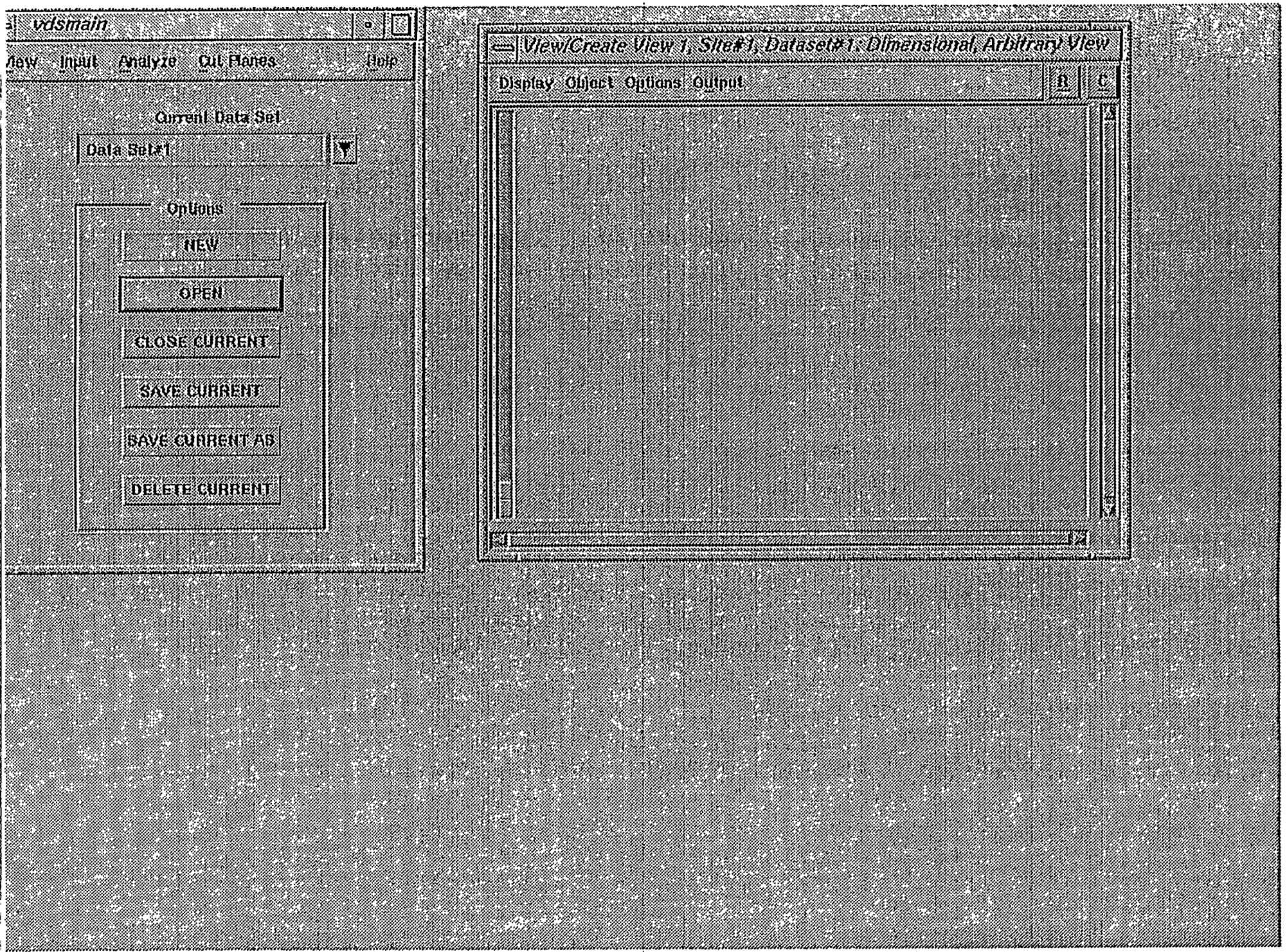


Figure 7-15 Visualization And Interaction CSC OOA Diagram


vdsmain
View Input Analyze Out Names Help
Current Data Set
Data Set#1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
View/Create View 1, Site#1, Dataset#1, Dimensional, Arbitrary View
Display Object Options Output
B
C

(16a) (16b)

The menus on the VDS View Window (Display, Object, Options, and Output) allow the operator to control the appearance of the data within the view window; activate an object control panel for interacting with the geometric objects associated with the view window's dataset; tailor the view options for a particular view window; and output hardcopies of the volumetric, property, and object databases. Figure 7-16b contains an example of a VDS View Window.

7.9.1.3 View Defaults Dialogs

Initial values for the VDS View Window control parameters come from a disk file (view.prm). A separate parameter file exists for each dataset and the parameters for each dataset may be customize using the View Defaults Dialogs. One dialog (refer to Figure 7-17) permits tailoring of all non-color related parameters. The other dialog (refer to Figure 7-18) allows tailoring of the view window coloring.

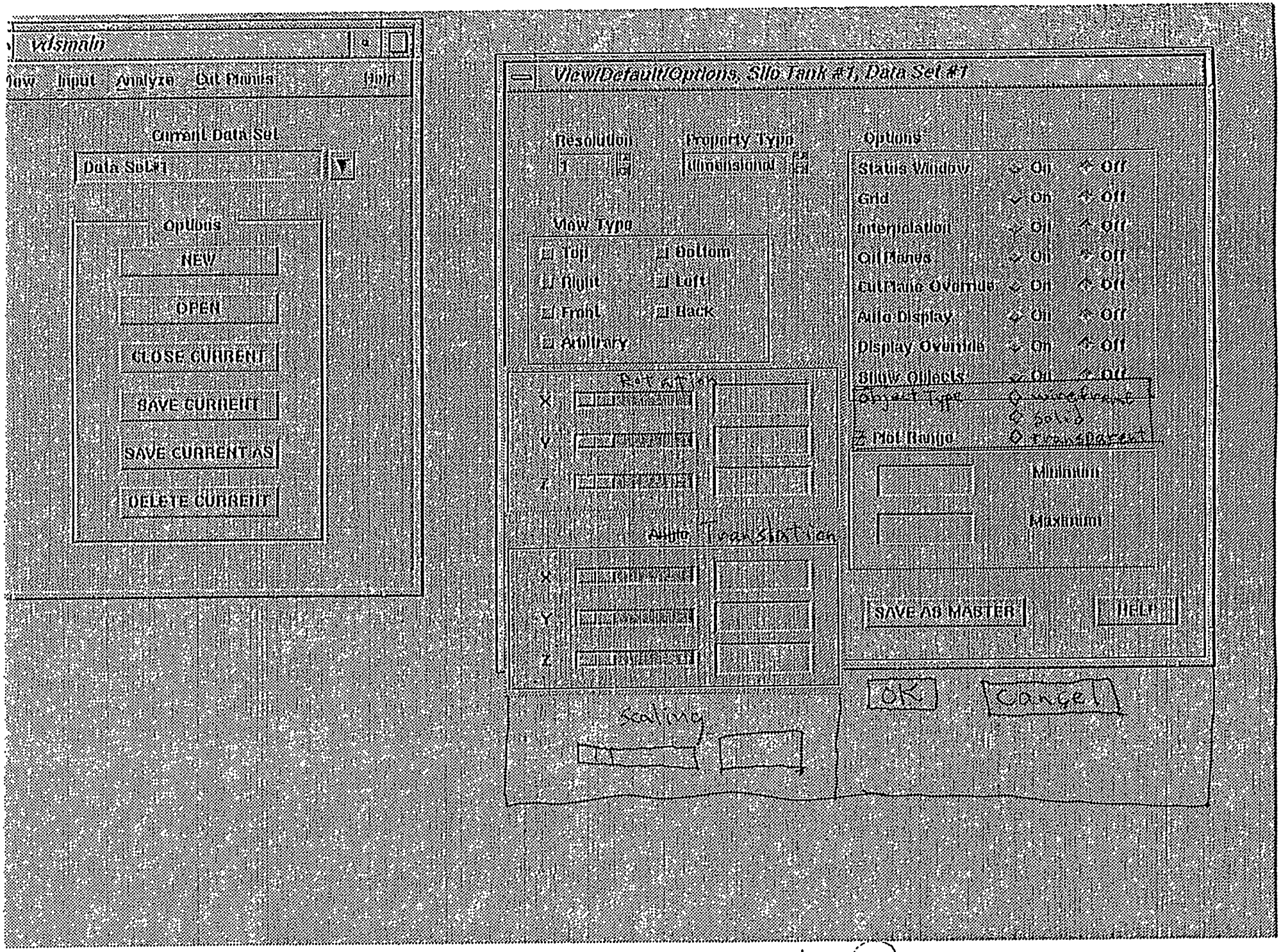
When activated from the Volumetric Data Window VIEW menu (Defaults button), the dataset default parameters can be edited. Each dialog supports a SAVE AS MASTER button that allows the current settings to saved as the system Master View Parameters File (etc/viewprms.def) which become the initial settings for all new datasets. When activated from the VDS View Window DISPLAY menu, the in-memory copy for a particular view window can be edited. The SAVE AS MASTER button does not appear.

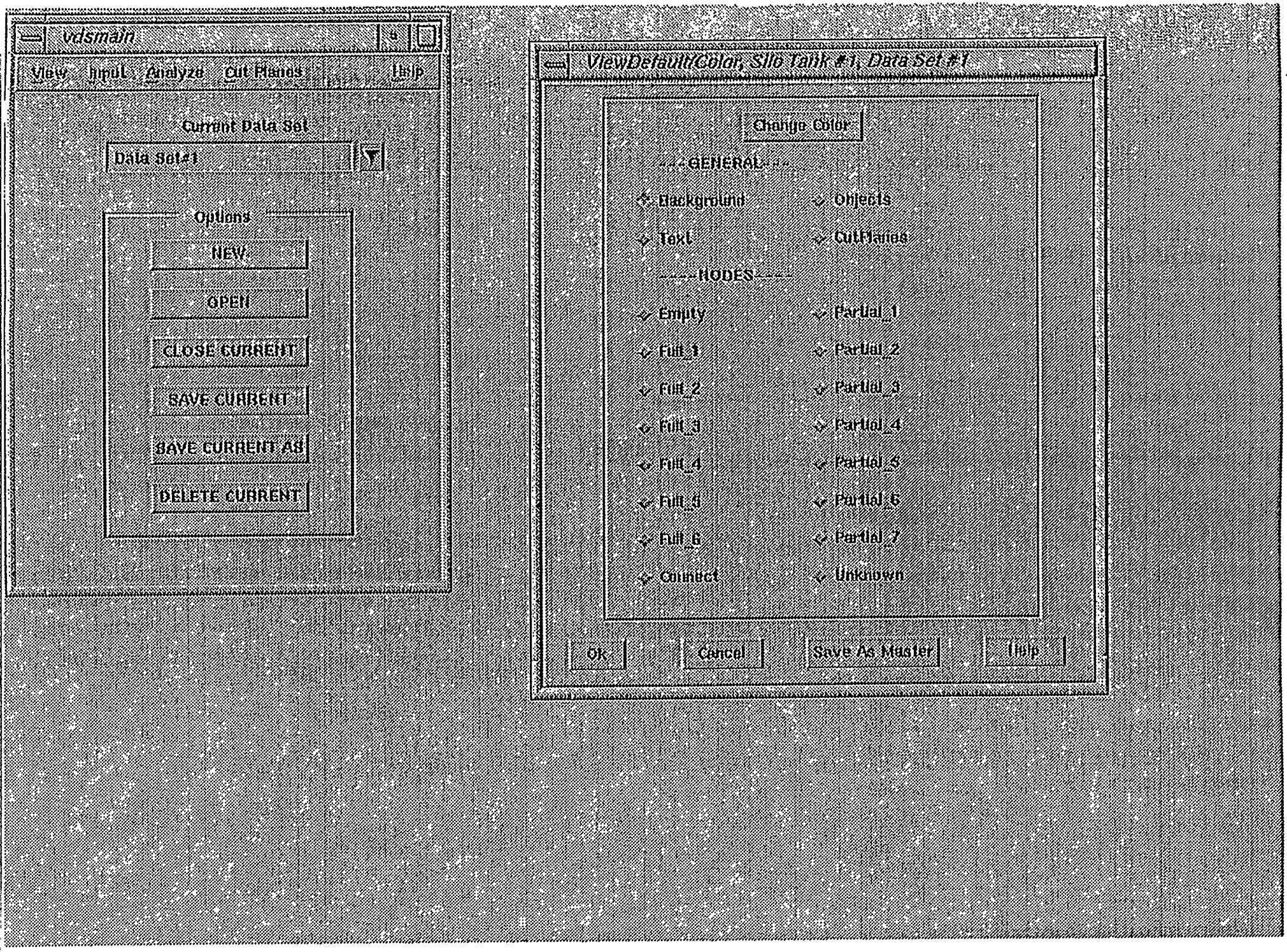
7.9.1.4 View WindowTransform Dialog

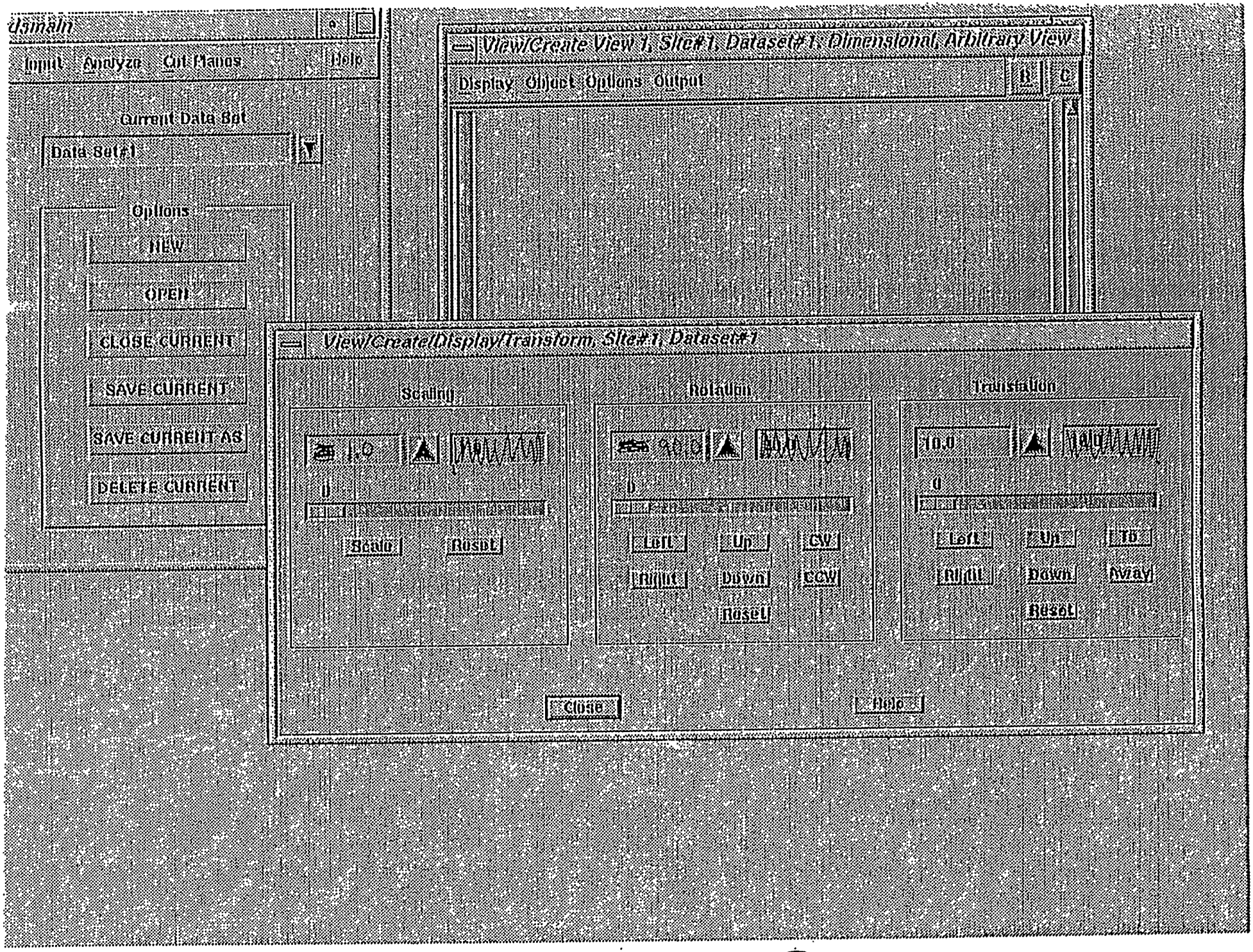
The View Window Transform Dialog is activated from the VDS View Window DISPLAY menu and provides the functionality for rotating, translating, and scaling the data within a view window. These functions are performed via sliders, edit fields, and buttons. Transforming the view can be perfromed by either moving the slider bar to the appropriate position, by selecting one of the predefined values in the pop-down selection dialog, or by entering the value manually into the edit field. The value contained in the edit field will be applied to the view as soon as one of the push buttons are clicked. These buttons will scale, rotate, or translate the view accordingly. Only arbitrary views can be rotated. This window is an alternative to using the view window's scaling and translation sliders located on the sides and bottom of the view window; and the rotation cube accessible from the menu bar "R" button. Figure 7-19 contains an example of this window.

7.9.1.5 Geometric Object Dialogs

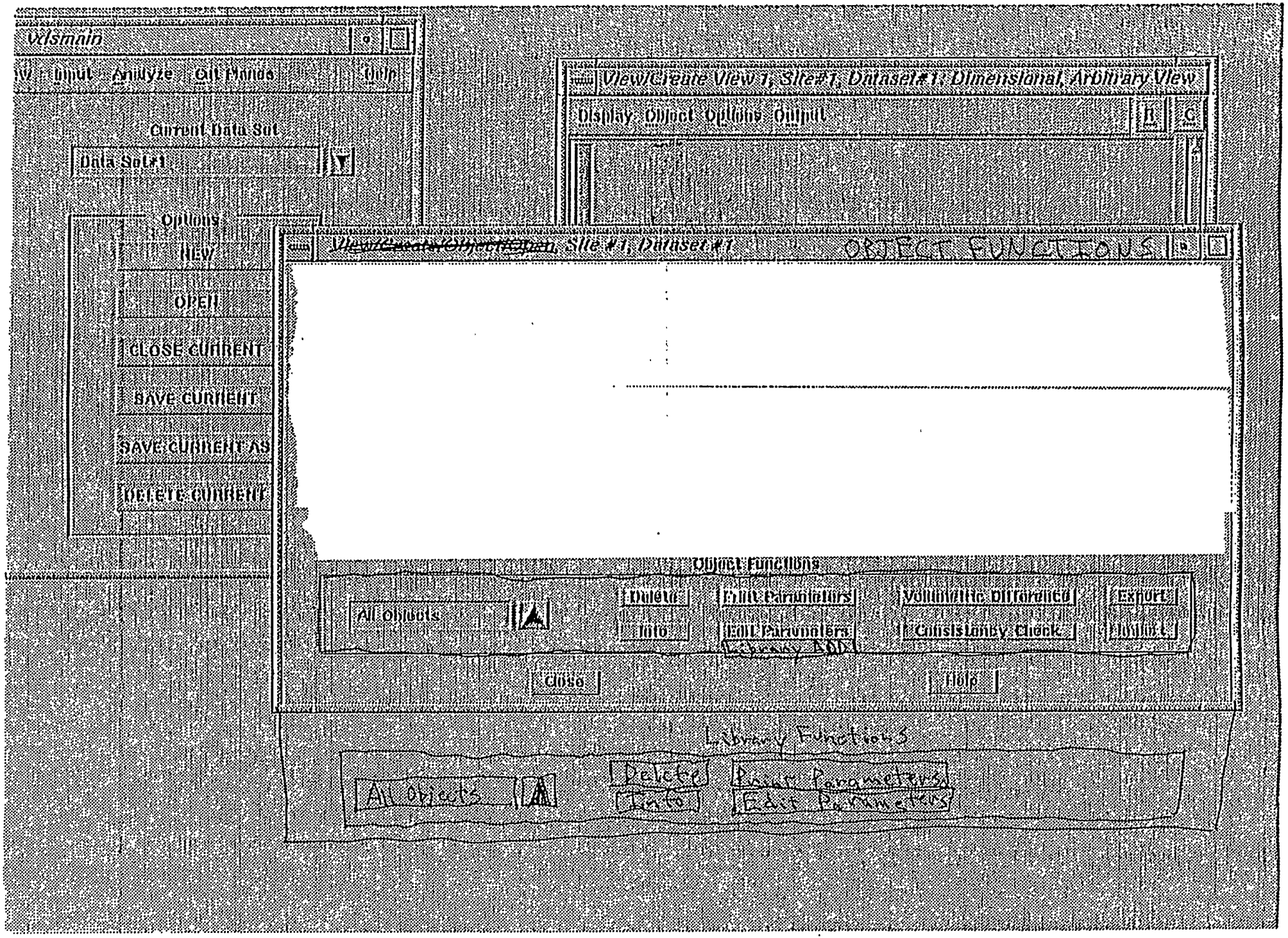
The Geometric Object Dialogs provide the capability to create and edit geometric objects. These dialogs consist of a functions dialog (Figure 7-20a) and an editing dialog (Figure 7-20b) which can bring up an object edit sub-window (Figure 7-21) and/or an object face reshaping sub-window (Figure 7-22). The functions dialog allows the operator to perform functions on objects (delete, print, export, etc.), as well as functions on library objects. The editing dialog allows the operator to create new objects, and bring up object editing and resizing sub-windows.The object editing sub-window allows the operator to scale, translate, and rotate a selected object or face of the object. The reshaping window allows the operator to reshape a face of the object. The Geometric Object Functions and Editing Dialogs are activated from the VDS View Window OBJECT menu.

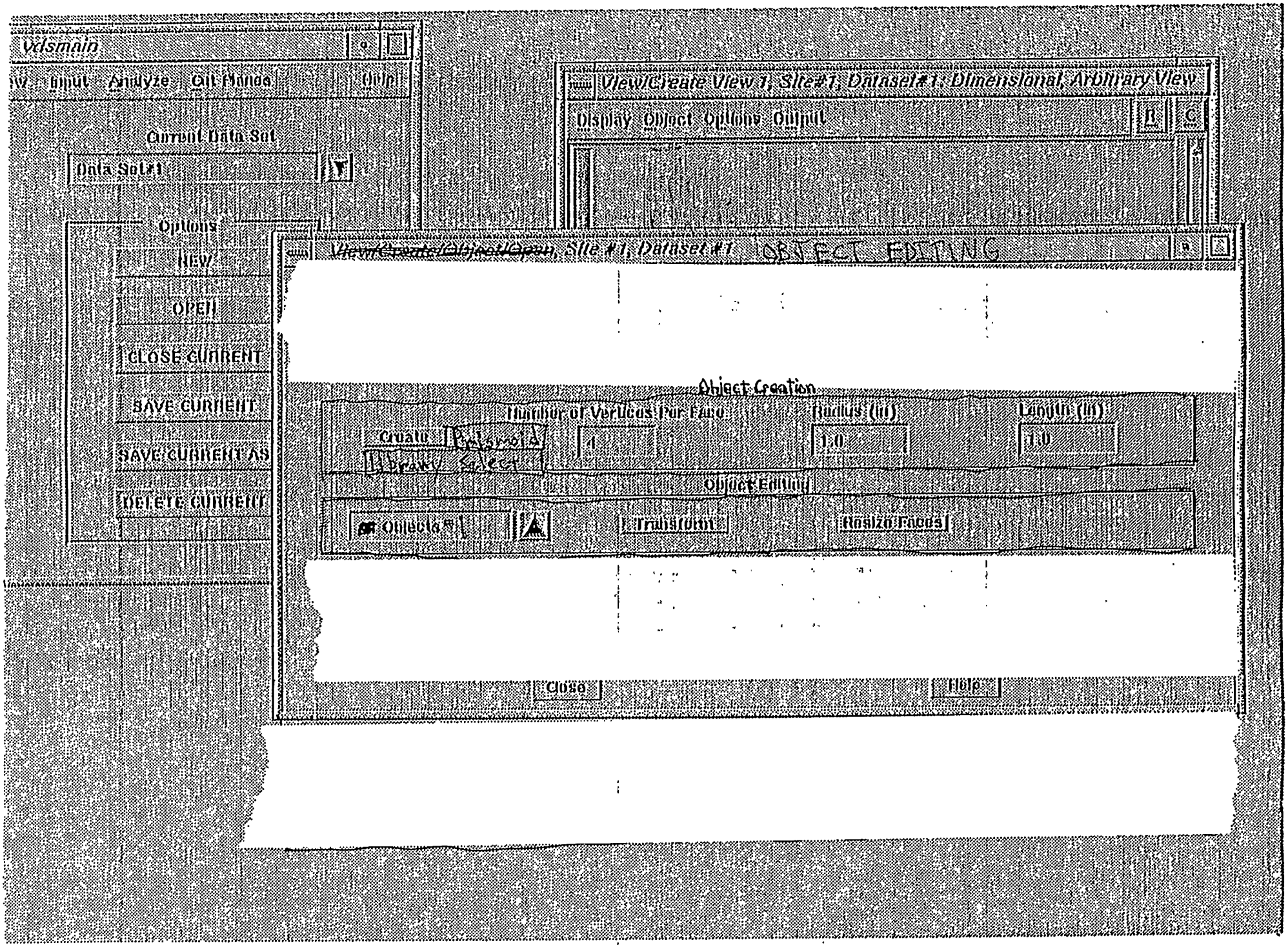

vdsmain
View Input Analyze Edit Menus Help
Current Data Set
Data Set #1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
View/Default/Options, Silo Tank #1, Data Set #1
Resolution
1
Property Type
Unconstrained
View Type
Top
Bottom
Right
Left
Front
Back
Arbitrary
Edit Origin
X
Y
Z
Auto Translation
X
Y
Z
Scaling
Options
Status Window On Off
Grid On Off
Interpolation On Off
Cut Planes On Off
Cut Plane Override On Off
Auto Display On Off
Display Override On Off
Show Objects On Off
Object Type
Plot Range
Minimum
Maximum
SAVE AS MASTER
HELP
OK
Cancel

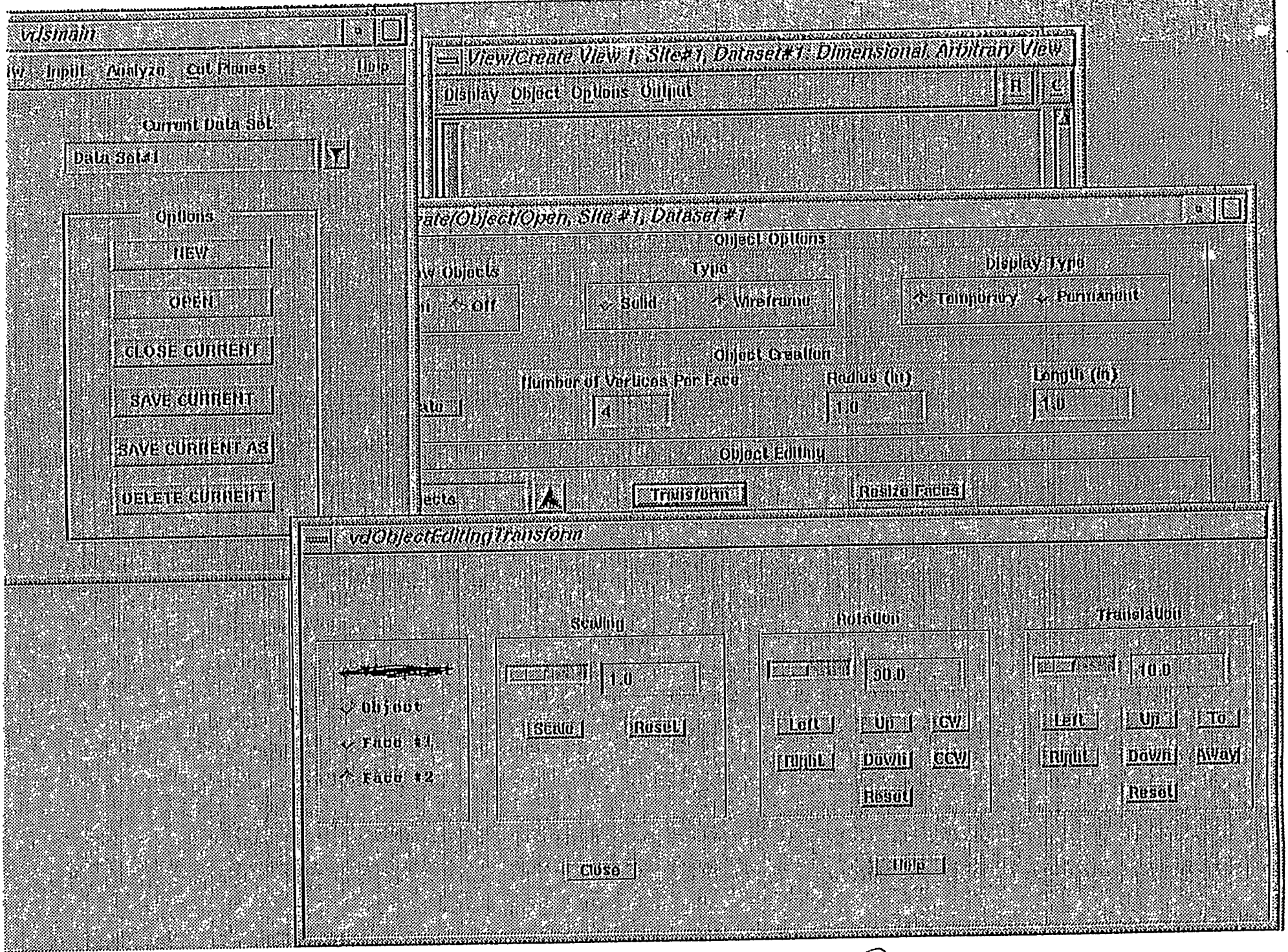

vdsmain
View
Input
Analyze
Cut Planes
Help
Current Data Set
Data Set#1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
View/Default/Color, Silo Tank #1, Data Set #1
Change Color
---GENERAL---
Background
Text
Objects
CutPlanes
---NODES---
Empty
Full_1
Full_2
Full_3
Full_4
Full_5
Full_6
Connect
Partial_1
Partial_2
Partial_3
Partial_4
Partial_5
Partial_6
Partial_7
Unknown
OK
Cancel
Save As Master
Help


Usimain
Input Analyze Cut Planes Help
Current Data Set
Data Set#1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
View/Create View 1, Site#1, Dataset#1: Dimensional, Arbitrary View
Display Object Options Output
View/Create/Display/Transform, Site#1, Dataset#1
Scaling
1.0
Scale
Reset
Rotation
Left
Up
CW
Right
Down
CCW
Reset
Translation
10.0
Left
Up
To
Right
Down
Away
Reset
Close
Help

7.9 1 & 1 (19)


vlsmain
Input Analyze GUI Modes Help
Current Data Set
Data Set#1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
View/Create View 1, Site#1, Dataset#1: Dimensional Arbitrary View
Display Object Options Output
R
C
View/Create/Object/Open Site #1, Dataset #1
OBJECT FUNCTIONS
Object Functions
All Objects
Delete
Info
Edit Parameters
Edit Parameters
Voxel-wise Difference
Consistency Check
Export
Duplicate
Legacy ADL
Close
Help
Library Functions
All Objects
Delete
Info
Print Parameters
Edit Parameters


vdsmain
Input
Analyze
Gui Manage
Output
Current Data Set
Data Set#1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
View/Create View 1, Site#1, Dataset#1, Dimensional, Arbitrary View
Display
Object
Options
Output
View/Create/Object/Open, Site #1, Dataset#1
OBJECT EDITING
Object Creation
Create
Number of Vertices Per Face
Radius (in)
1.0
Length (in)
1.0
Object Editing
Objects
Translate
Resize Faces
Close
Help


vdsmain
Input
Analyze
cut Planes
Help
Current Data Set
Data Set #1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
View/Create View 1, Site#1, Dataset#1, Dimensional Arbitrary View
Display
Object
Options
Output
Create/Object/Open, Site #1, Dataset #1
Object Options
Type
Solid
Wireframe
Display Type
Temporary
Permanent
Off
Object Creation
Number of Vertices Per Face
4
Radius (in)
1.0
Length (in)
1.0
Object Editing
Transform
Resize Faces
vdObjectEditingTransform
Object
Face #1
Face #2
Scaling
1.0
Scale
Reset
Rotation
90.0
Left
Up
CW
Right
Down
CCW
Reset
Help
Close
Translation
10.0
Left
Up
To
Right
Down
Away
Reset

7017-2 (21)

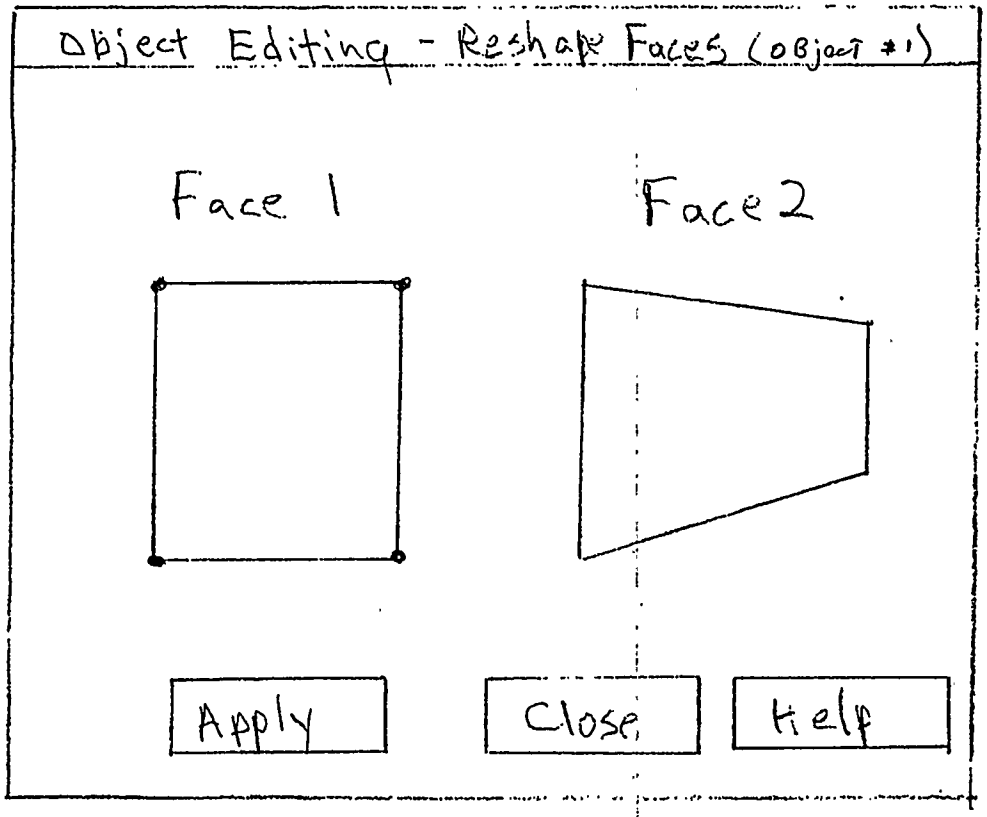

Object Editing - Reshape Faces (object #1)
Face 1
Face 2
Apply
Close
Help

fig 7.9.1.7-3 (22)

7.9.1.6 Data Input Dialogs

The Data Input Dialogs are activated from the Volumetric Data Window INPUT menu and provide a means to enter data into the Volumetric Data Subsystem. There is a separate dialog for each data input function: Change Region (Figure 7-23), Add Point(Figure 7-24), Erase Point (Figure 7-25), and Add File (Figure 7-26).

7.9.1.7 Analyze Dialogs

The Analyze Dialogs are activated from the ANALYZE menu on the Volumetric Data Window. The dialogs allow either a volumetric difference between two datasets or a 2.5D surface map of the data for up to two datasets. The results of the analyze functions are displayed in a variation of the VDS View Window. The results view window can be manipulated: scaled, translated, or rotated (arbitrary view only). The results view window can also be cutplaned or sectioned. The major difference between an analyze results window and a VDS view window is the title label and the inability to define or view objects in the analyze results window. Also, since analyze results windows aren't considered real views, they can't be saved, restored, or copied. Figures 7-27 and 2-28 contain examples of the volumetric difference and 2.5D surface map analyze dialogs.

7.9.1.8 Cutplane Dialogs

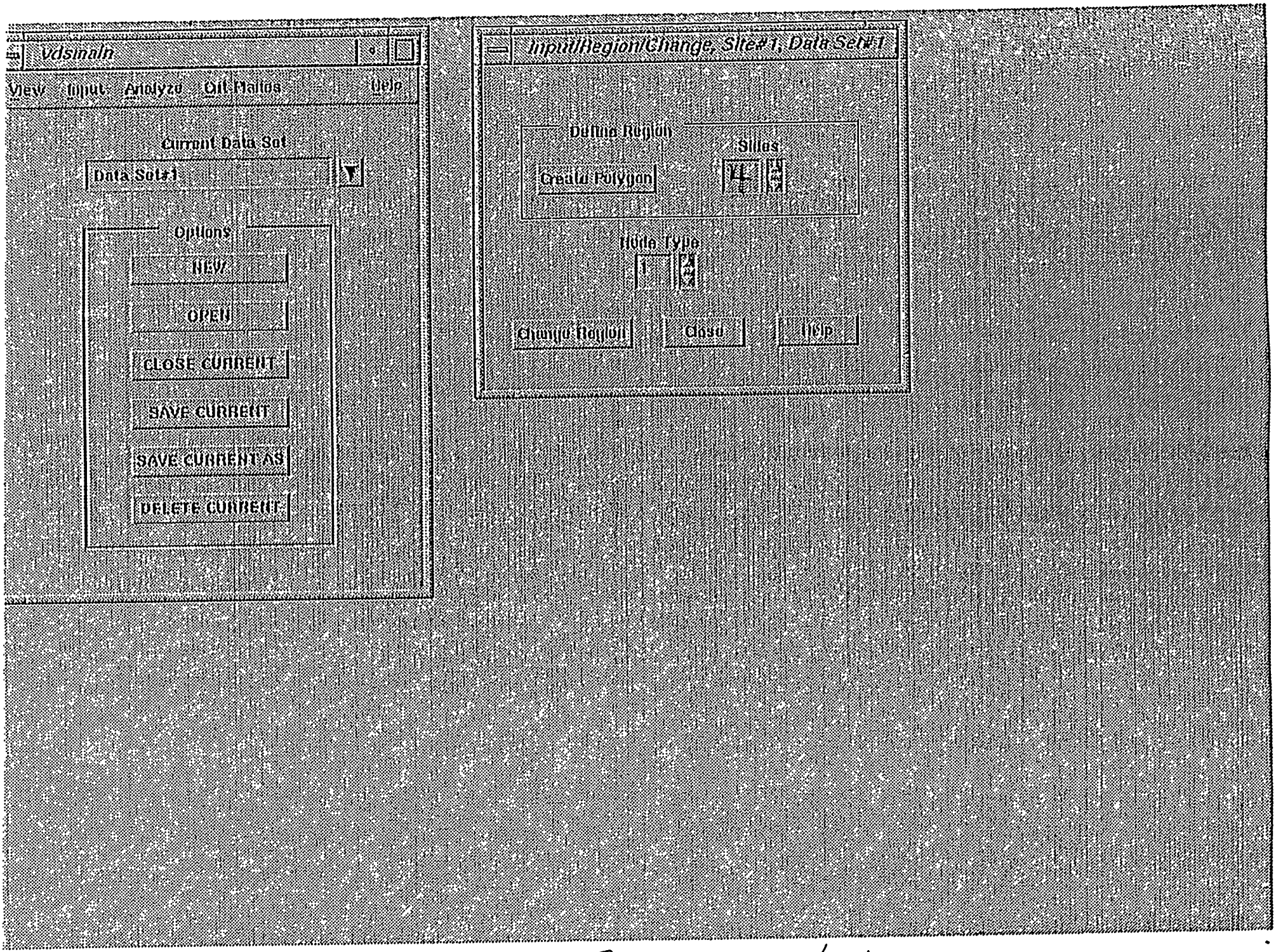
The Cutplane Dialogs are activated from the CUTPLANE-Edit button on the Volumetric Data Window or from the "C" button on the VDS View Window menubar. The dialogs provide the capability for defining and viewing cutplanes in a dataset. These dialogs consist of an editing screen screen (or selection screen, depending on from where the dialog was activated (Figure 7-29), and display windows for a selected view. The editing (or selection) screen allows the operator to select the cutplane to be redefined, enable/disable cutplanes, and define the cutplane's position/orientation.. If activated from the VDS View Window Menubar, four types of cutplane windows can be displayed from the main cutplane selection window:

1. A cutplane cube window which allows the operator to redefine a cutplane and then apply it to the current dataset. Refer to Figure 7-30 for an example of this control window.

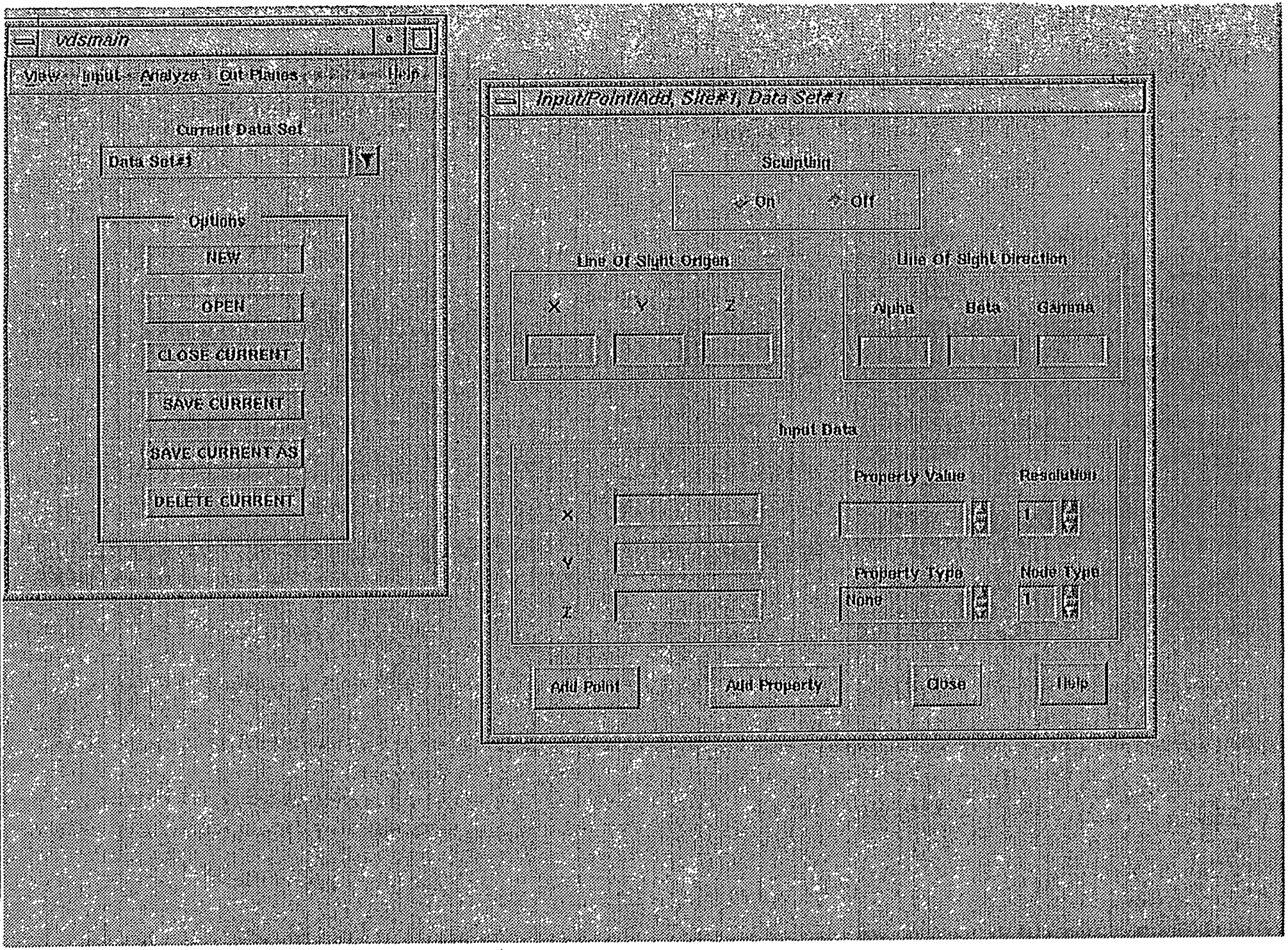
2. A section cube window which allows the operator to redefine a section and then apply it to the current view. A section is a single cutplane which has some width associated with it. Only the data contained within the section will be displayed. This section can then be selected and dragged to the desired location where the view will be updated to reflect the latest position. Refer to Figure 7-30 for an example of this control window.

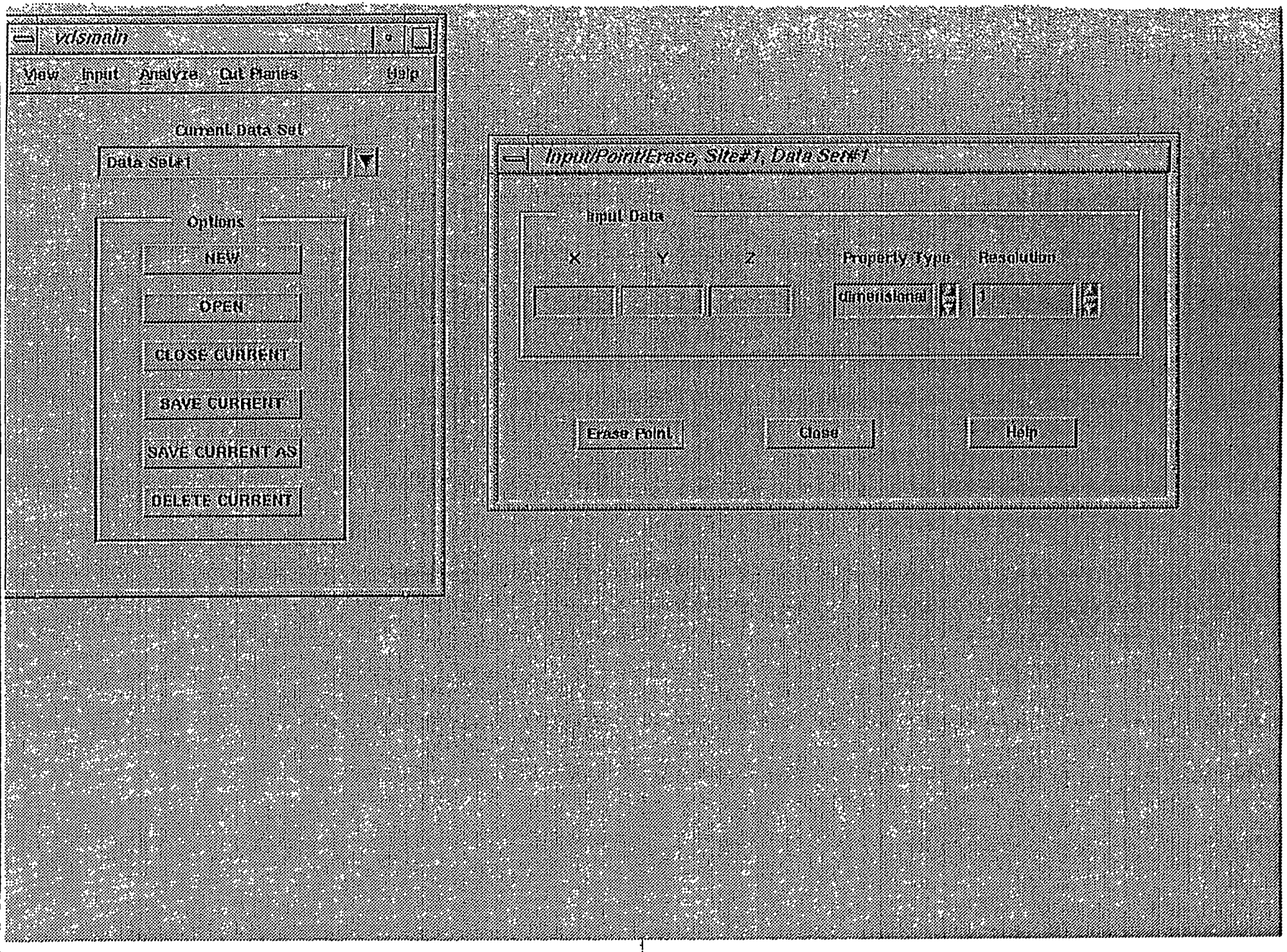
3. A profile window which allows the operator to view a horizontal or vertical slice of the data in a specific view. Refer to Figure 7-36 for an example of this window.

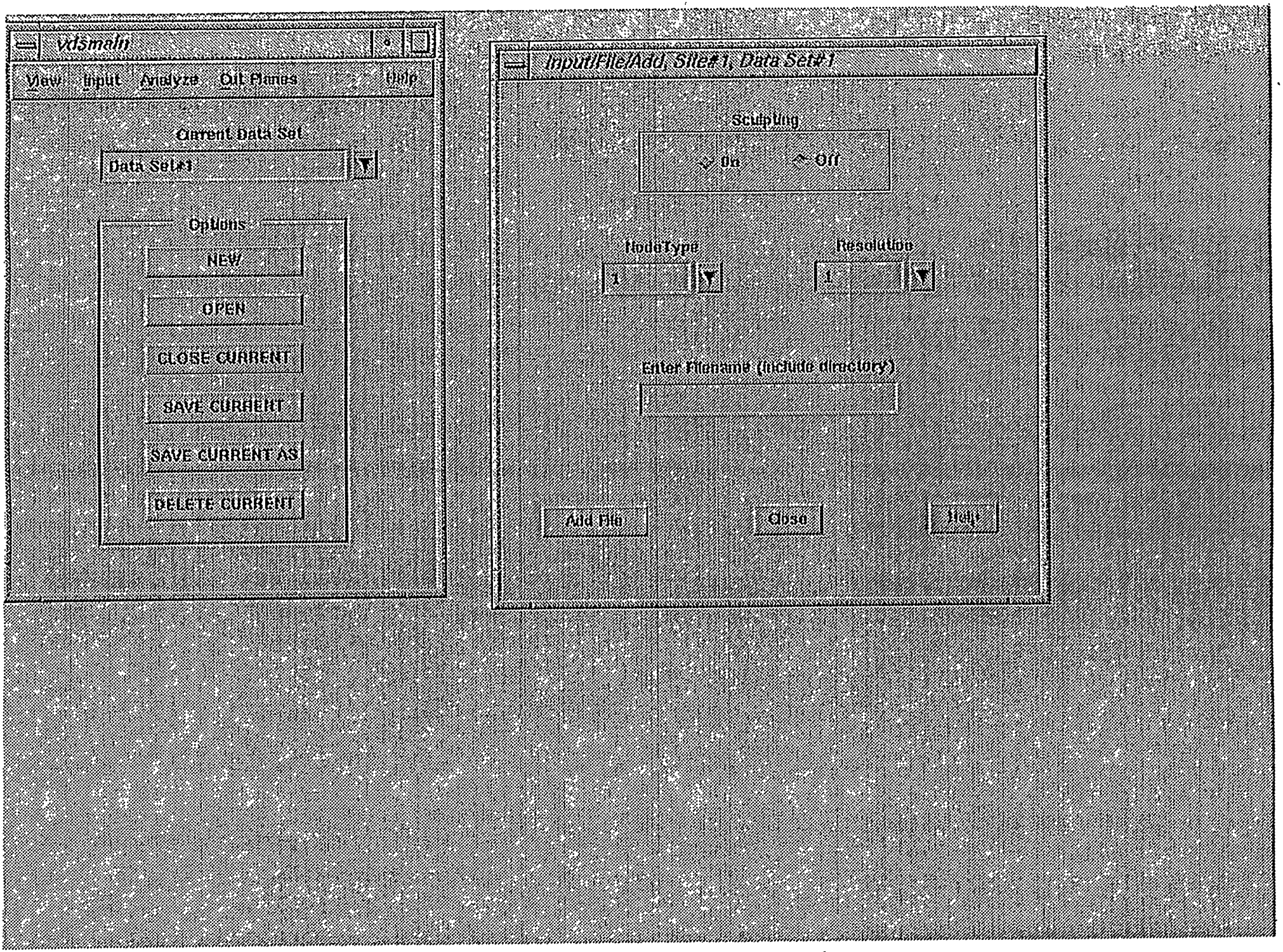
4. A slice window which allows the operator to view an arbitrary slice of the data in a specific view. Refer to Figure 7-37 for an example of this window.

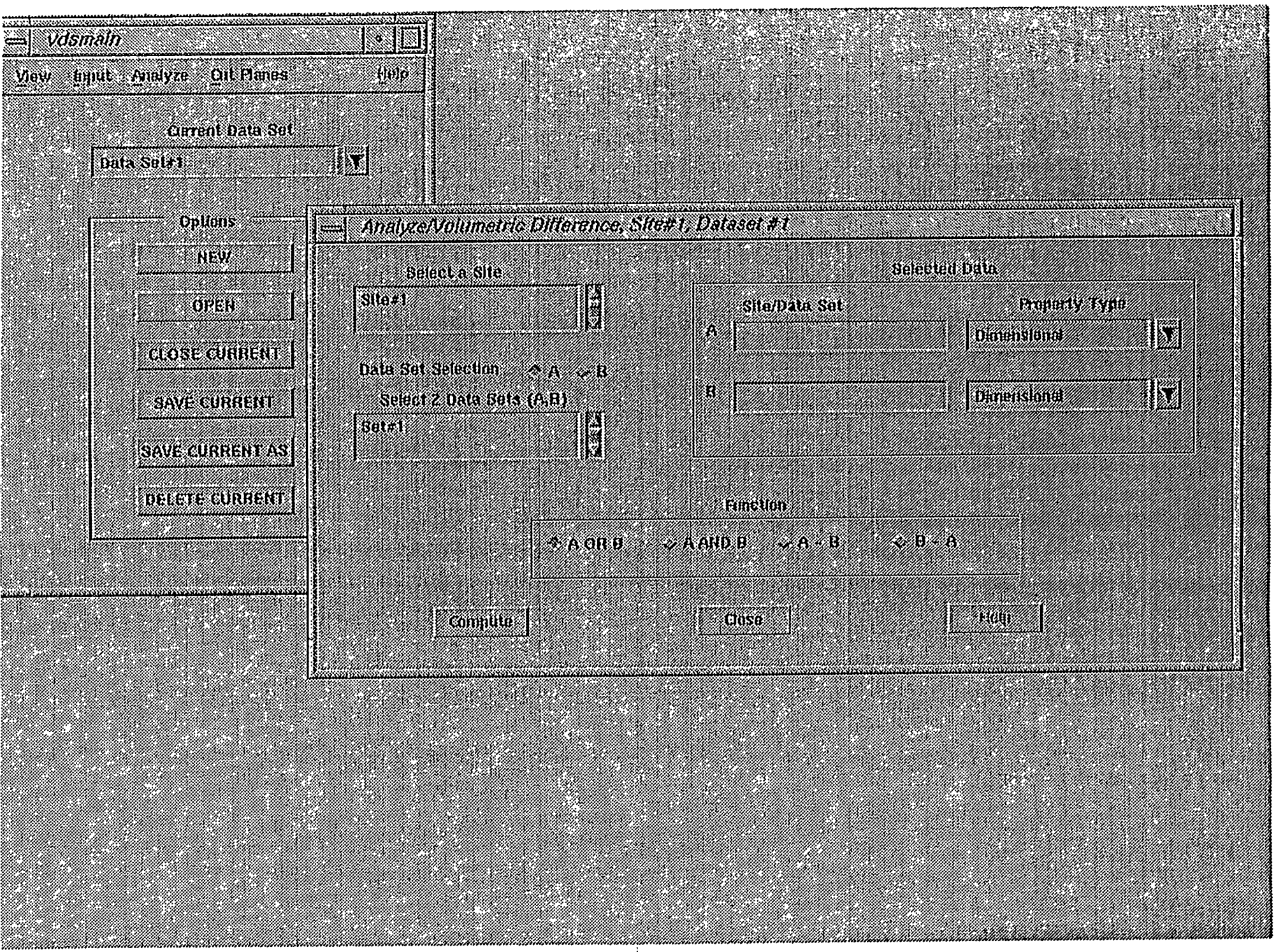

Vdsmain
View
Input
Analyze
Edit Matlab
Help
Current Data Set
Data Set#1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
Input/Region/Change, Site#1, Data Set#1
Define Region
Create Polygon
Sites
Node Type
1
2
3
Change Region
Close
Help

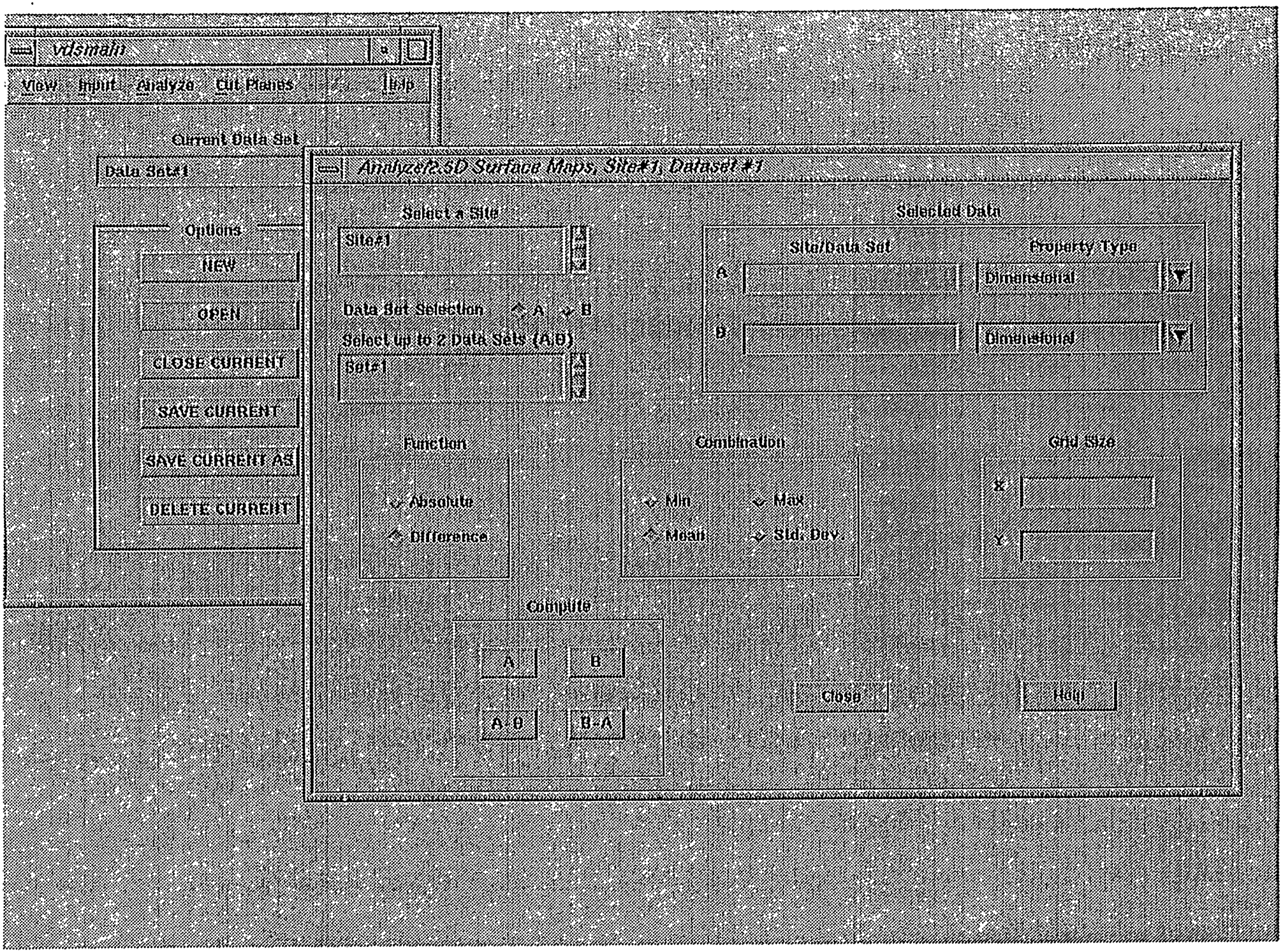
7.9 1 3-1 (23)

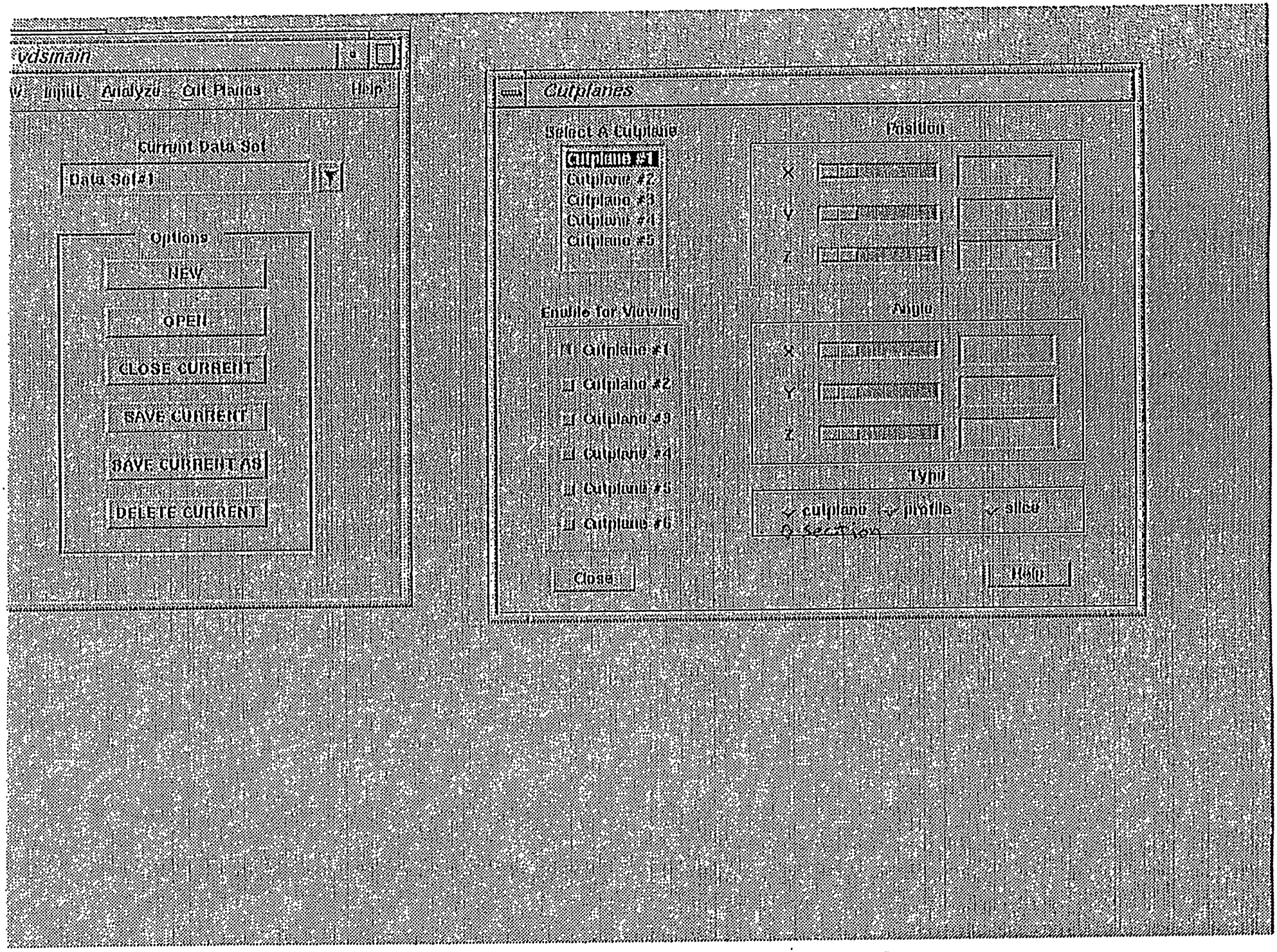

vdsmain
View
Input
Analyze
Out Planes
Help
Current Data Set
Data Set#1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
Input/Point/Add, Site#1, Data Set#1
Sculpting
On
Off
Line Of Sight Origin
X
Y
Z
Line Of Sight Direction
Alpha
Beta
Gamma
Input Data
X
Y
Z
Property Value
Resolution
1
Property Type
None
Node Type
1
Add Point
Add Property
Close
Help

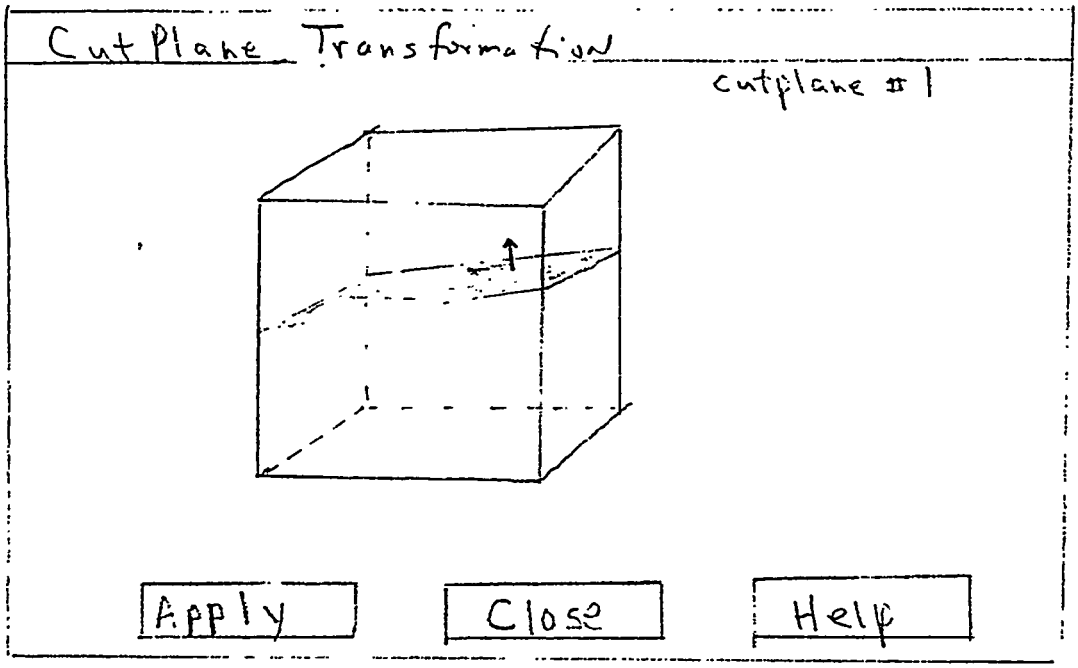

vdsmain
View
Input
Analyze
Quit Names
Help
Current Data Set
Data Set#1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
Input/Point/Erase, Site#1, Data Set#1
Input Data
X
Y
Z
Property Type
Resolution
dimensional
1
Erase Point
Close
Help


vdsmain
View
Input
Analyze
Out Flames
Help
Current Data Set
Data Set#1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
Input/File/Add, Site#1, Data Set#1
Sculpting
On
Off
NodeType
1
Resolution
1
Enter Filename (include directory)
Add File
Close
Help


Vdsmain
View
Input
Analyze
Cut Planes
Help
Current Data Set
Data Set#1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
Analyze/Volumetric Difference, Site#1, Dataset #1
Select a Site
Site#1
Data Set Selection
A
B
Select 2 Data Sets (A,B)
Set#1
Selected Data
Site/Data Set
Property Type
A
Dimensional
B
Dimensional
Function
A OR B
A AND B
A - B
B - A
Compute
Close
Help


vdsmain
View
Input
Analyze
Cut Planes
Help
Current Data Set
Data Set#1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
Analyze/2.5D Surface Maps, Site#1, Dataset#1
Select a Site
Site#1
Data Set Selection
A
B
Select up to 2 Data Sets (A,B)
Set#1
Selected Data
Site/Data Set
Property Type
A
Dimensional
B
Dimensional
Function
Absolute
Difference
Combination
Min
Max
Mean
Std. Dev.
Grid Size
X
Y
Compute
A
B
A-B
B-A
Close
Help


vdsmain
Analyze
Cut Planes
Help
Current Data Set
Data Set #1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
Cutplanes
Select A Cutplane
Cutplane #1
Cutplane #2
Cutplane #3
Cutplane #4
Cutplane #5
Enable for Viewing
Cutplane #1
Cutplane #2
Cutplane #3
Cutplane #4
Cutplane #5
Cutplane #6
Close
Position
X
Y
Z
Apply
X
Y
Z
Type
cutplane
probe
slice
Help


Cut Plane Transformation
cutplane # 1
Apply
Close
Help

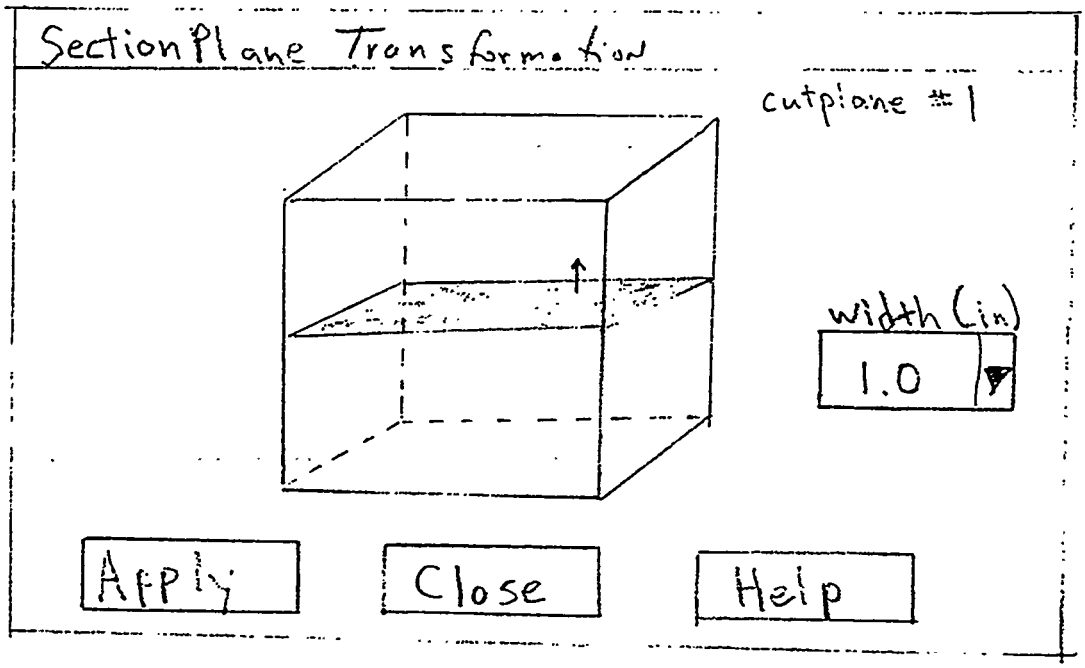

Section Plane Transformation
cutplane # 1
width (in)
1.0
Apply
Close
Help

fig - 3.1.b-2

7.9.2. Interface To VDS Server

Functionally, this CSC is a client of the VDS Server. It consists of a CVolumetricDataClient object which encapsulates the client-side interface to the VDS Server. All requests to perform non-user interface functions are channeled through the CVolumetricDataClient class, which accepts requests, formats VDS commands, transmits the commands to the VDS server for processing, then waits for a reply from the server.

7.9.3. Class Descriptions

The Visualization and Interaction CSC software is objected oriented, implemented in C++ and consists of approximately 49 classes. This section identifies the CSC classes and discusses their general characteristics (attributes, behavior and relationships). Section 7.10.3 will discuss the major functions assigned to the CSC and describe how the software classes implement the functions.

In keeping with good object-oriented analysis practices, classes are grouped by the categories Problem Domain Component (PDC) and Human Interface Component (HIC).

7.9.3.1 Problem Domain Classes

There are no Problem Domain Component classes for the Visualization and Interaction CSC. However, the HIC classes for this CSC interact with PDC objects from the other CSCs, including::

CCutplane	CSetOfCutplanes
CVdsDataset	CSetOfVdsDatasets
CGeometric3dObject	CSetOfGeometric3dObjects
CMaterialProperty	CSetOfMaterialProperties
CSite	CSetOfSites
CVdsView	CSetOfVdsViews
CVolumetricDataClient	

7.9.3.2 Human Interface Classes

The HIC classes are broken into two functional groups: the VDS Main Window classes and the VDS View Window classes. Refer to Figure 7-31 and 7-32 for OOA/OOD diagrams of the interactions among these classes. The classes for the Visualization and Interaction CSC are identified and described below. Much of the layout and code for these classes will be generated by the UNIRAS UIMX tool.

7.9.3.2.1 VDS Main Window Classes

VdsGlobalHIC: Class contains all the global data definitions for the VDS HIC classes.

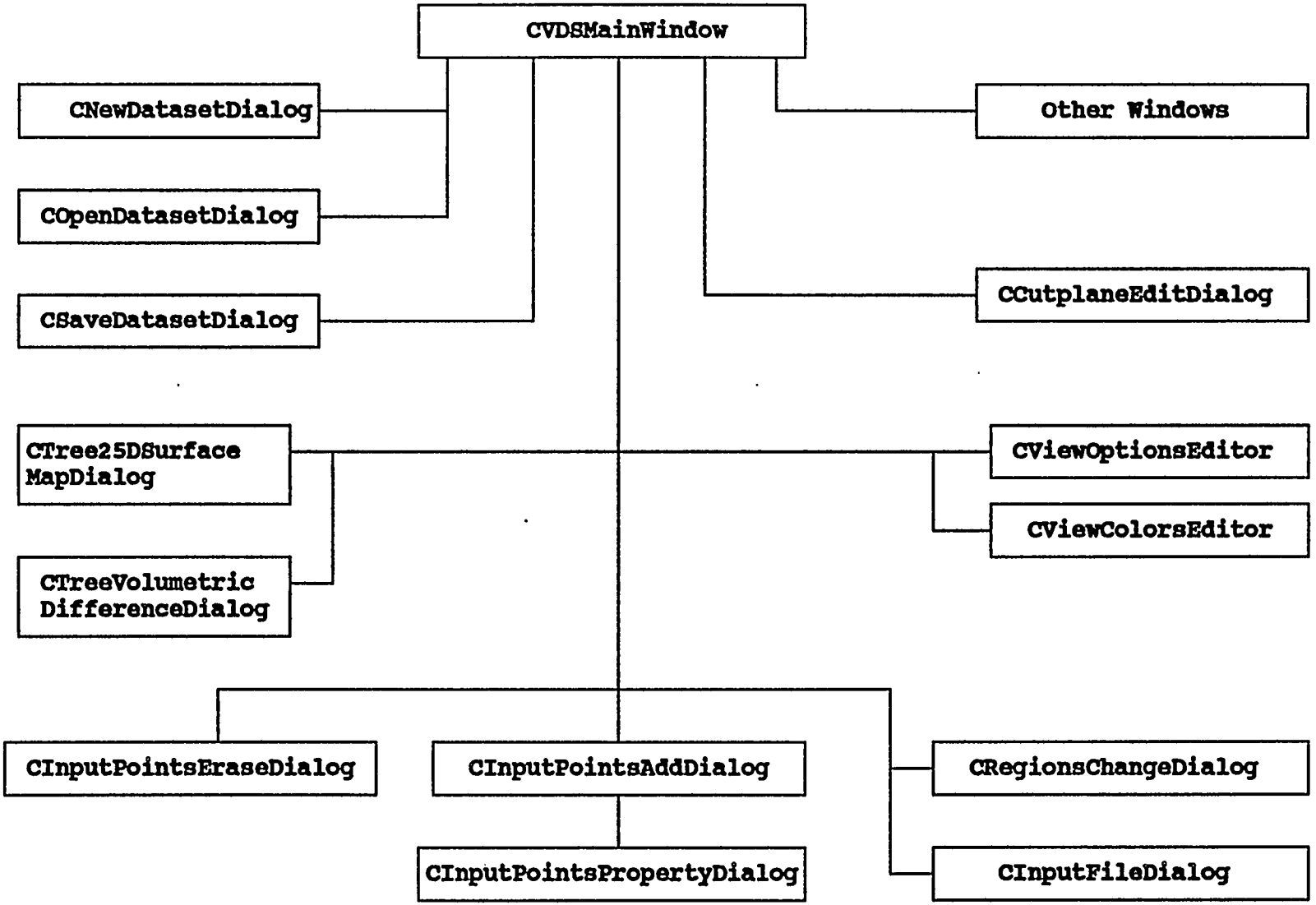


Figure 7-31 VDS Main Window Classes

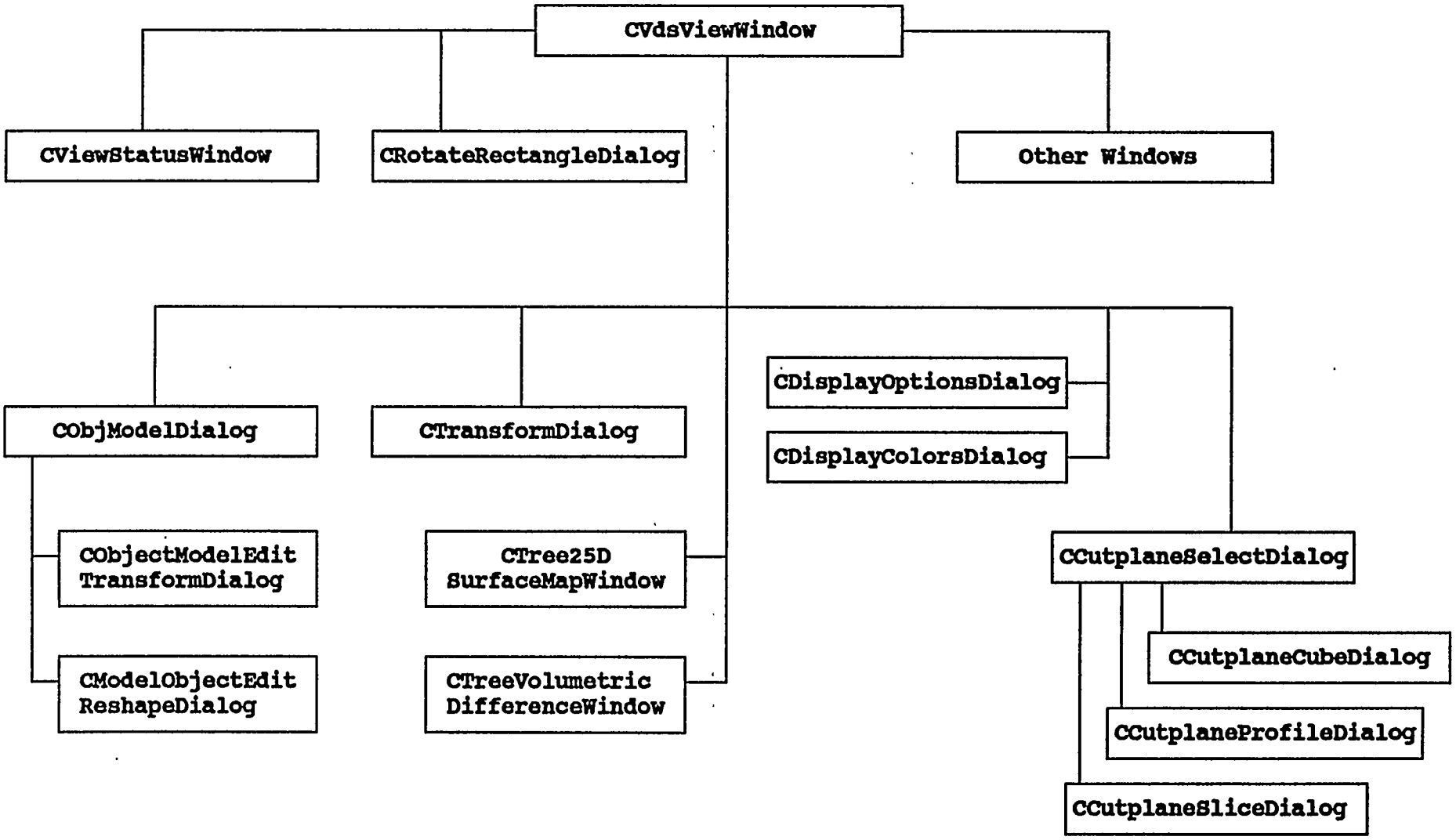


Figure 7-32 VDS View Window Classes

CCloseDataSetVerifyDialog: Class to display a dataset close message for the operator to verify. It is instantiated when the CLOSE button is pressed from the CVdsMainWindow. YES, NO, HELP buttons are supported.

CCloseViewsVerifyDialog: Class to display a view close message for the operator to verify. It is instantiated when the VIEW-Close button is pressed from the CVdsMainWindow menubar. YES, NO, HELP buttons are supported.

CCutplaneEditDialog: Class to display a dialog that allows the operator to edit the positions and orientations of the six predefined cutplanes for the currently selected dataset. The dialog contains a selection list of all six cutplanes for the operator to select the one to be edited. This dialog is displayed when the CUTPLANE-Edit button is pressed from the CVdsMainWindow menubar. OK, CLOSE, and HELP buttons are supported. Refer Figure 7-33 for an example of this dialog.

CDefaultViewColorsEditor: Class to display a control panel that allows the operator to change the default view colors for the currently selected dataset. These colors will pertain to all views for the current dataset. The operator can save the edited changes as the new master default file in the \etc directory which will become permanent for subsequent sessions. This dialog is displayed when the VIEW-Default Color button is pressed from the CVdsMainWindow menubar. OK, CANCEL, and HELP buttons are supported. Refer to Figure 7-18 for an example of this dialog window.

CDefaultViewOptionsEditor: Class to display a control panel that allows the operator to change any default view options for the currently selected dataset. These options will pertain to all subsequent new views for the current dataset. The operator can save the edited changes as the new master default file in the \etc directory which will become permanent for subsequent sessions. This dialog is displayed when the VIEW-Default Options button is pressed from the CVdsMainWindow menubar. OK, CANCEL, HELP buttons are supported. Refer to Figure 7-17 for an example of this dialog window.

CDeleteDataSetVerifyDialog: Class to display a dataset deletion message for the operator to verify. It is instantiated when the DELETE CURRENT button is pressed from the CVdsMainWindow YES, NO, HELP buttons are supported.

CInputFileDialog: Class to display a dialog that allows the operator to input an ASCII file of dimensional and possible property data The format of the file is. yet to be determined. The tree resolution, node type, and sculpting options must be entered before the file is read in and will apply to all points in the file as they are entered into the octree. This dialog is displayed when the INPUT-File Add button is pressed on the CVdsMainWindow menubar. ADD, CLOSE, and HELP buttons are supported. Refer to Figure 7-26 for an example of this dialog window.

CInputPointAddDialog: Class to display a dialog that allows the operator to input a data point and possible property value. If sculpting is selected, then the line of sight origin and direction must be input. In addition, the tree resolution, node type and property type must be entered. If the desired property type is not in the selection list, then the operator may add one with the ADD PROPERTY button. The property type must be consistent for all points in a given dataset and cannot be changed midstream. However, the node type, resolution, and sculpting can vary from point to point. This dialog is displayed when the INPUT-Point Add button is pressed on the CVdsMainWindow menubar. ADD, CLOSE, ADD PROPERTY, and HELP buttons are supported. Refer to Figure 7-24 for an example of this dialog window.

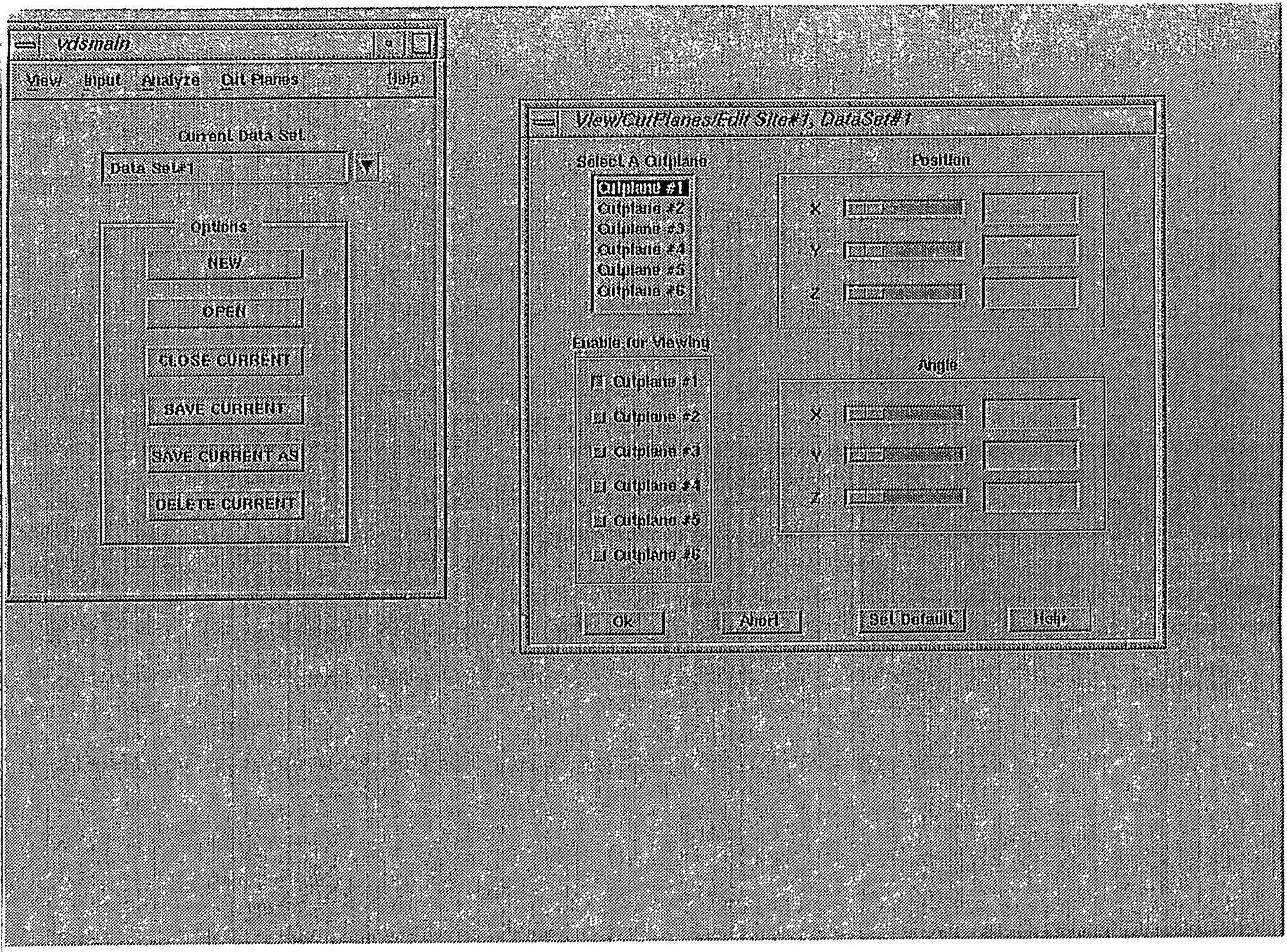
CInputPointEraseDialog: Class to display a dialog that allows the operator to erase a data point and possible property value. The resolution of the point must also be inputted to be able to find the exact node to be removed from the octree. This dialog is displayed when the INPUT-Point Erase button is pressed on the CVdsMainWindow menubar. ERASE, CLOSE, and HELP buttons are supported. Refer Figure 7-25 for an example of this dialog window.

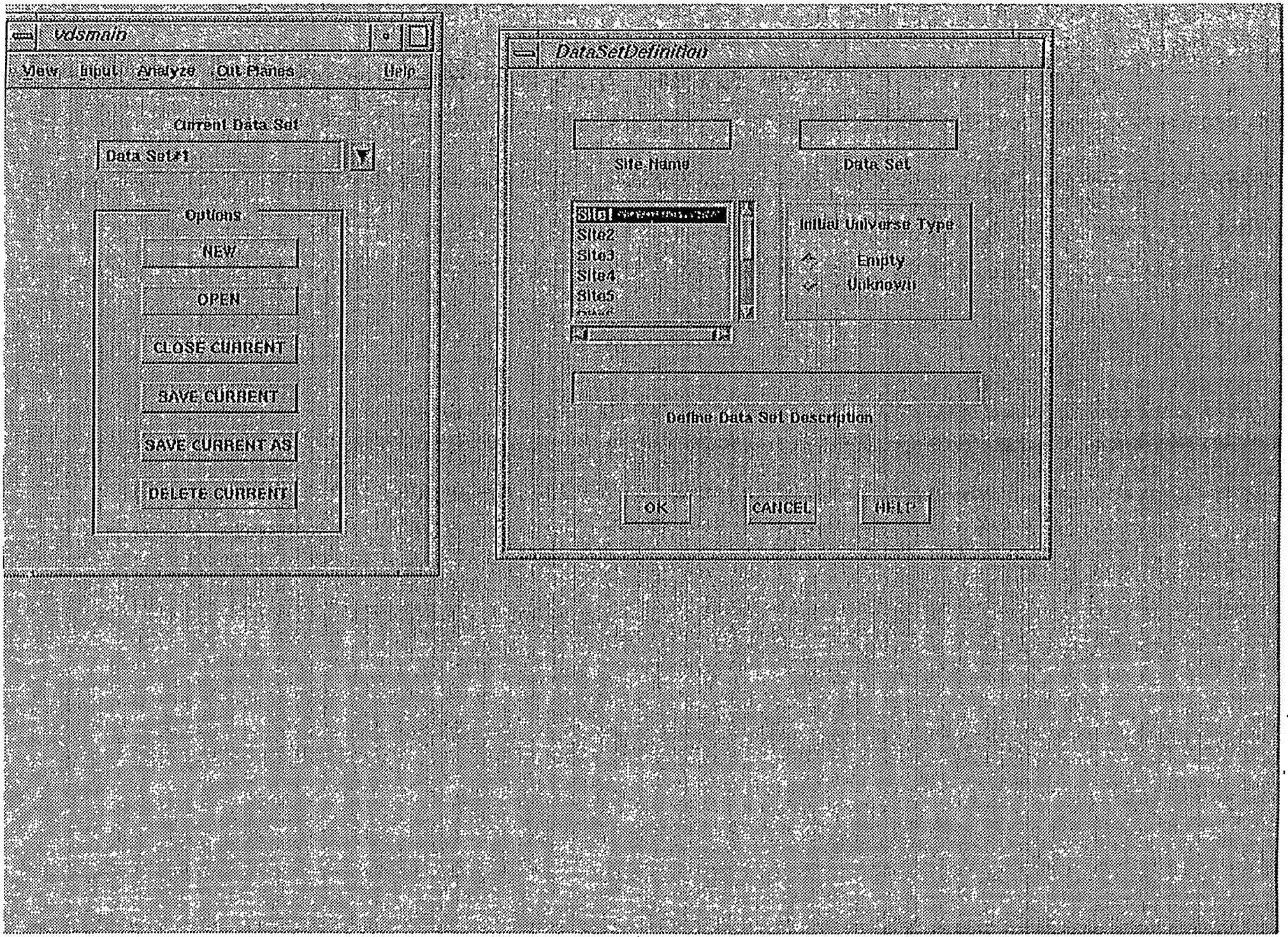
CInputPropertyDialog: Class to display a dialog that allows the operator to input a property value for the currently inputted data point. This is a separate dialog as opposed to a field on the input dialog because it is not known yet how complex the property values will be. This dialog is displayed when a Property Type other than "dimensional" is selected from the CInputPointAddDialog. OK, CANCEL, and HELP buttons are supported.

CInputRegionsChangeDialog: Class to display a dialog that allows the operator to define a region and to change the node type for the region (eg.. change full nodes to empty nodes) The region definition will be handled like the object definition is done. This dialog is displayed when the INPUT-Region Change button is pressed on the CVdsMainWindow menubar. CHANGE, CLOSE, and HELP buttons are supported. Refer to Figure 7-23 for an example of this dialog window.

CInputRegionsEraseDialog: Class to display a dialog that allows the operator to define a region and to erase all nodes in the region (eg. change all nodes to empty or unknown) This is grayed out in Phase II because the input regions change function can be used to accomplish the same purpose.

CNewDatasetDialog: Class to display a dialog that allows the operator to define a new dataset. The operator must define the site and dataset description as well as the type of universe that the data will start out as (empty or unknown). The dataset name is automatically generated and displayed. The type of property is not needed to be known until data is actually being stored in the new dataset. This dialog is instantiated when the NEW button is pressed from the CVdsMainWindow. OK, CANCEL, and HELP buttons are supported. See Figure 7-34 for an example of this dialog window.


vdsmain
View Input Analyze Cut Planes Help
Current Data Set
Data Set#1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
View/CutPlanes/Edit Site#1, DataSet#1
Select A Cutplane
Cutplane #1
Cutplane #2
Cutplane #3
Cutplane #4
Cutplane #5
Cutplane #6
Position
X
Y
Z
Enable for Viewing
Cutplane #1
Cutplane #2
Cutplane #3
Cutplane #4
Cutplane #5
Cutplane #6
Angle
X
Y
Z
OK
Abort
Set Default
Help


vdsmain
View
Input
Analyze
Out Planes
Help
Current Data Set
Data Set#1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
DataSetDefinition
Site Name
Data Set
Site1
Site2
Site3
Site4
Site5
Initial Universe Type
Empty
Unknown
Define Data Set Description
OK
CANCEL
HELP

COpenDatasetDialog: Class to display a dialog that allows the operator to select an existing dataset. The operator must select the site and dataset. The type of property is displayed automatically. It is instantiated when the OPEN button is pressed from the CVdsMainWindow. OK, CANCEL, and HELP buttons are supported. Refer to Figure 7-35 for an example of this dialog window.

CSaveDatasetSelectionListDialog: Class to display a dialog that allows the operator to define a new dataset for the purposes of saving an existing dataset. The operator must define the site and dataset description. The dataset name is automatically generated and displayed. The type of property is already known from the current dataset. It is instantiated when the SAVE AS button is pressed from the CVdsMainWindow. OK, CANCEL, and HELP buttons are supported.

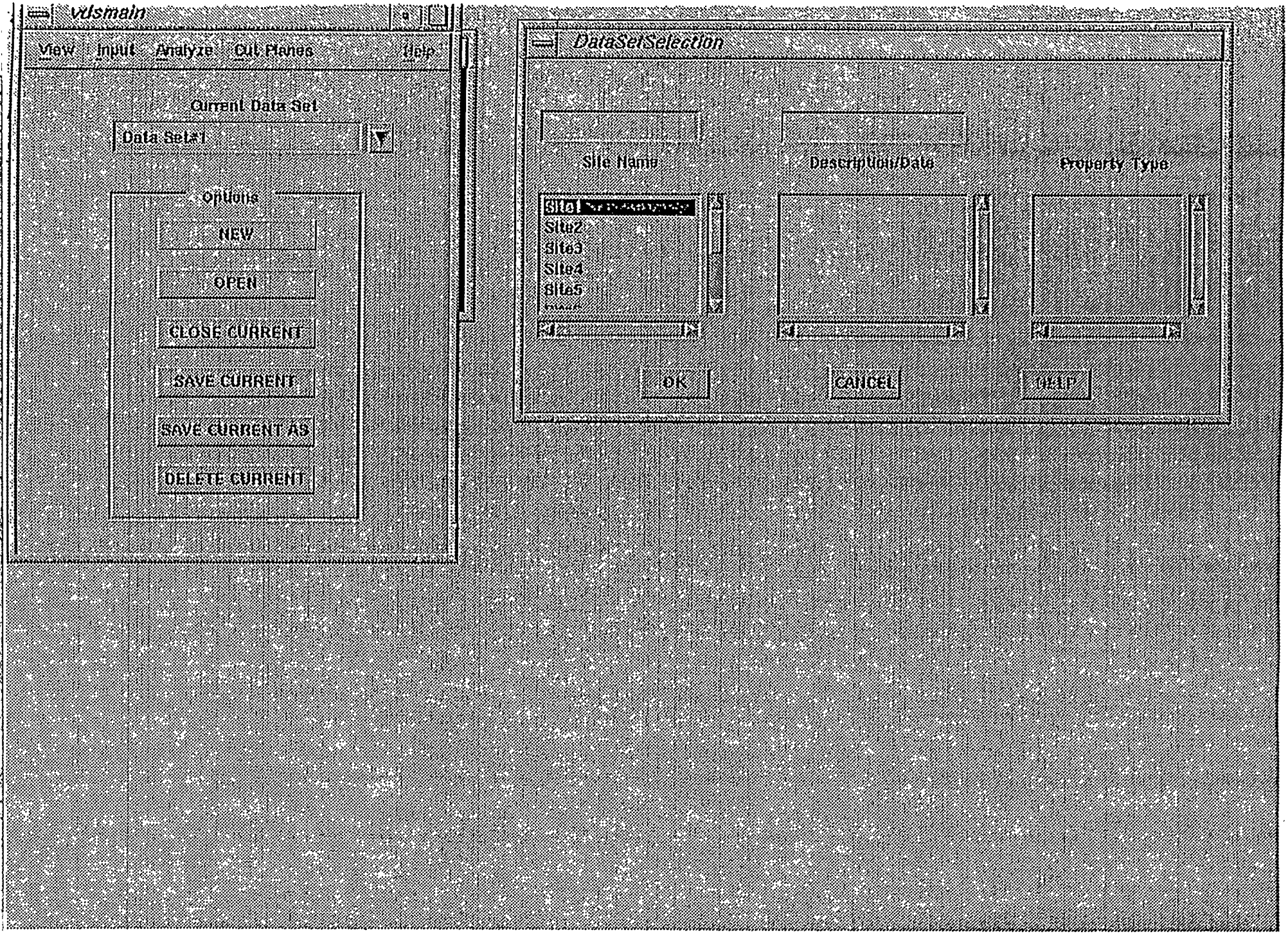
CSaveDatasetVerifyDialog: Class to display a "save changed dataset?" message for the operator to verify. It is instantiated when the CLOSE button is pressed from the CVdsMainWindow and the dataset has changed. YES, NO, HELP buttons are supported.

CSavedViewFileRequestNameDialog: Class to display a dialog that allows the operator to define a saved view file within the current site and dataset. Both the filename and description are required for operator input. This dialog is displayed when the VIEW-Save or VIEW-Save All buttons are pressed from the CVdsMainWindow menubar. OK, CANCEL, and HELP buttons are supported.

CSavedViewFileSelectionListDialog: Class to display a dialog that allows the operator to select an saved view file within the current site and dataset. Both the filename and description are displayed for operator selection This dialog is displayed when the VIEW-Restore button is pressed from the CVdsMainWindow menubar. OK, CANCEL, and HELP buttons are supported.

CSavedViewsVerifyDialog: Class to display a view save message for the operator to verify. It is instantiated when the Save or Save As button is pressed from the CVdsMainWindow. YES, NO, HELP buttons are supported.

CTree25DSurfaceMapDialog: Class to display the 2.5D surface map control panel that allows the operator to define up to two datasets to be analyzed and the function to be performed on the dataset(s). For Phase II we must select the two datasets within the same site. This dialog is displayed when the View/Analyze/2.5D Surface button is pressed from the CVdsMainWindow menubar. A, B, A-B, B-A, CLOSE, HELP buttons are supported. Refer to Figure 7-28 for an example of this dialog window.


vtksmain
View
Input
Analyze
Cut Planes
Help
Current Data Set
Data Set#1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
DataSetSelection
Site Name
Description/Data
Property Type
Site1
Site2
Site3
Site4
Site5
OK
CANCEL
HELP

CTreeVolumetricDifferenceDialog: Class to display the volumetric difference control panel that allows the operator to define the two datasets to be analyzed and the function to be performed on the datasets. For Phase II we must select the two datasets within the same site. This dialog is displayed when the View/Analyze/Volumetric Difference button is pressed from the CVdsMainWindow menubar. COMPUTE, CLOSE, HELP buttons are supported. Refer to Figure 7-27 for an example of this dialog window.

CVdsMainWindow: Class to display the VDS main window and handle all its callbacks. Typically the callbacks make calls to a CVolumetricDataClient object to send commands to the VDS Server program to perform PDC type functions. It is instantiated from the VDS user interface program. Refer to Figure 7-15 for an example of this window.

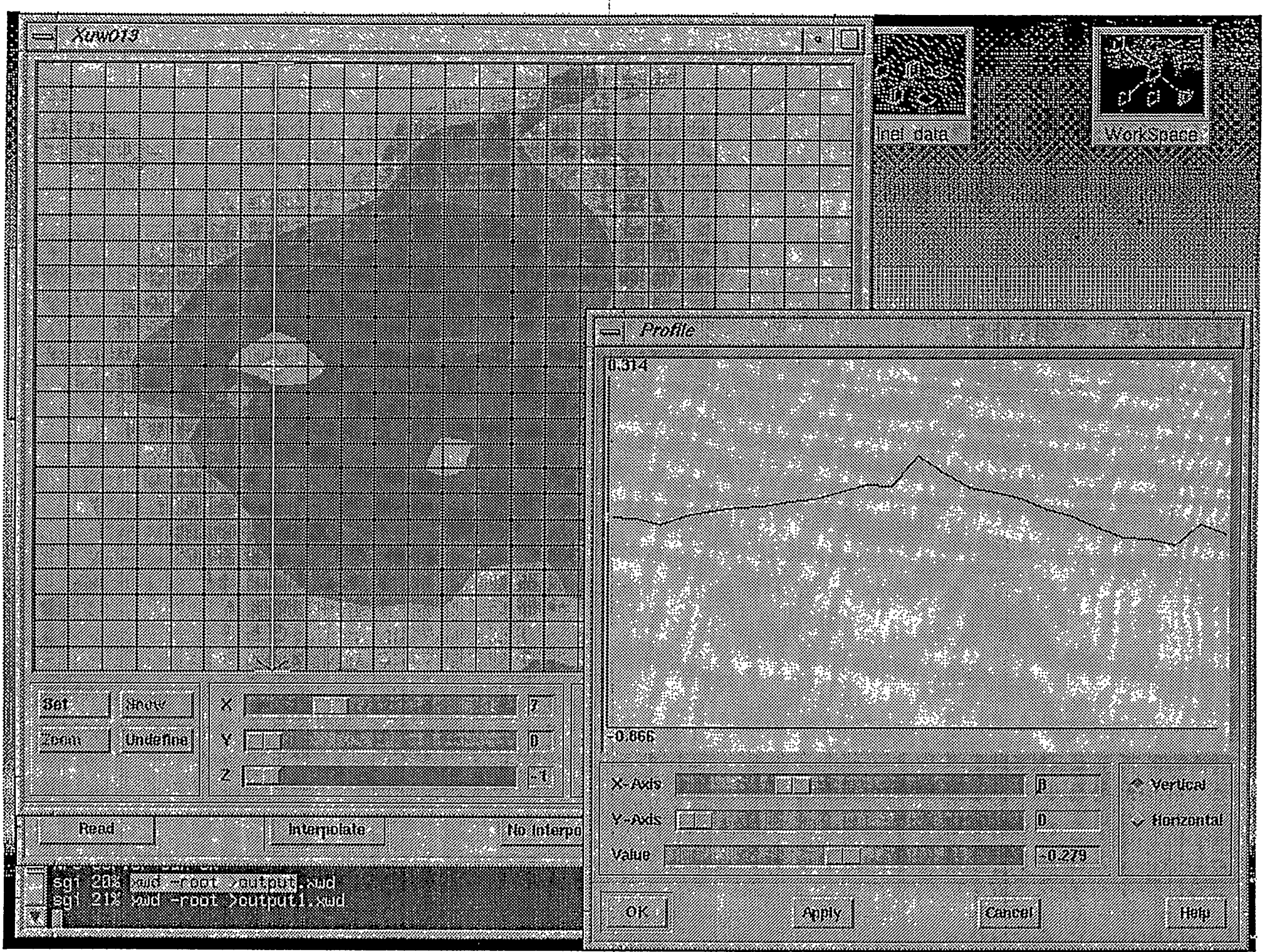
CViewSelectionListDialog: Class to display a dialog that allows the operator to select a view from a list of currently displayed views. OK, CANCEL, HELP buttons are supported.

7.9.3.2.2 VDS View Window Classes

CCutplaneCubeDialog: Class to display a dialog that allows the operator to edit the selected cutplane's position within the view. The cutplane can be repositioned in the cube using the middle mouse button or via sliders. An arrow is displayed on the cutplane to show sidedness (side of the plane on which the cutting will occur). This dialog is displayed when the Cutplane or Section button is pressed from the CCutplaneSelectDialog. CLOSE, and HELP buttons are supported. Refer to Figure 7-30 for an example of this window.

CCutplaneProfileDialog: Class to display a dialog that allows the operator to view a horizontal or vertical slice of the data in the current view. This dialog is displayed when the Profile button is pressed from the CCutplaneSelectDialog. CLOSE, and HELP buttons are supported. Refer to Figure 7-36 for an example of this window.

CCutPlaneSelectDialog: Class to display a dialog that allows the operator to select one of the six predefined cutplanes for redefinition. The dialog contains a selection list of all six cutplanes for the operator to select the one to be redefined. In addition, each cutplane can be individually enabled or disabled for viewing. Each cutplane's position and angle can be redefined via the edit fields, or they can be redefined visually by pressing the Cutplane or Section button which brings up a CCutplaneCubeDialog. This dialog is displayed when the "C" (Cutplane) button is pressed from the VDS View Window menubar. CLOSE, and HELP buttons are supported. Refer to Figure 7-29 for an example of this dialog window.


Xwd013
Inel_data
WorkSpace
Profile
0.314
-0.866
Set
Index
Zoom
Undefine
X
Y
Z
7
0
-1
Read
Interpolate
No Interpo
sgi 20% xwd -root >output.xwd
sgi 21% xwd -root >output1.xwd
X-Axis
Y-Axis
Value
0
0
-0.279
Vertical
Horizontal
OK
Apply
Cancel
Help

Profile (36)

CCutplaneSliceDialog: Class to display a dialog that allows the operator to view an arbitrary slice of the data in the current view. This dialog is displayed when the Slice button is pressed from the CCutplaneSelectDialog. CLOSE, and HELP buttons are supported. Refer to Figure 7-37 for an example of this dialog window.

CDisplayColorsDialog: Class to display a color control panel that allows the operator to change any view color for the currently displayed view. It is instantiated when the DISPLAY-Colors menu button is pressed from the CVdsViewWindow menubar. These colors will pertain to only the view in which the colors option window was invoked. APPLY, CLOSE, and HELP buttons are supported. Refer to Figure 7-38 for an example of this dialog window.

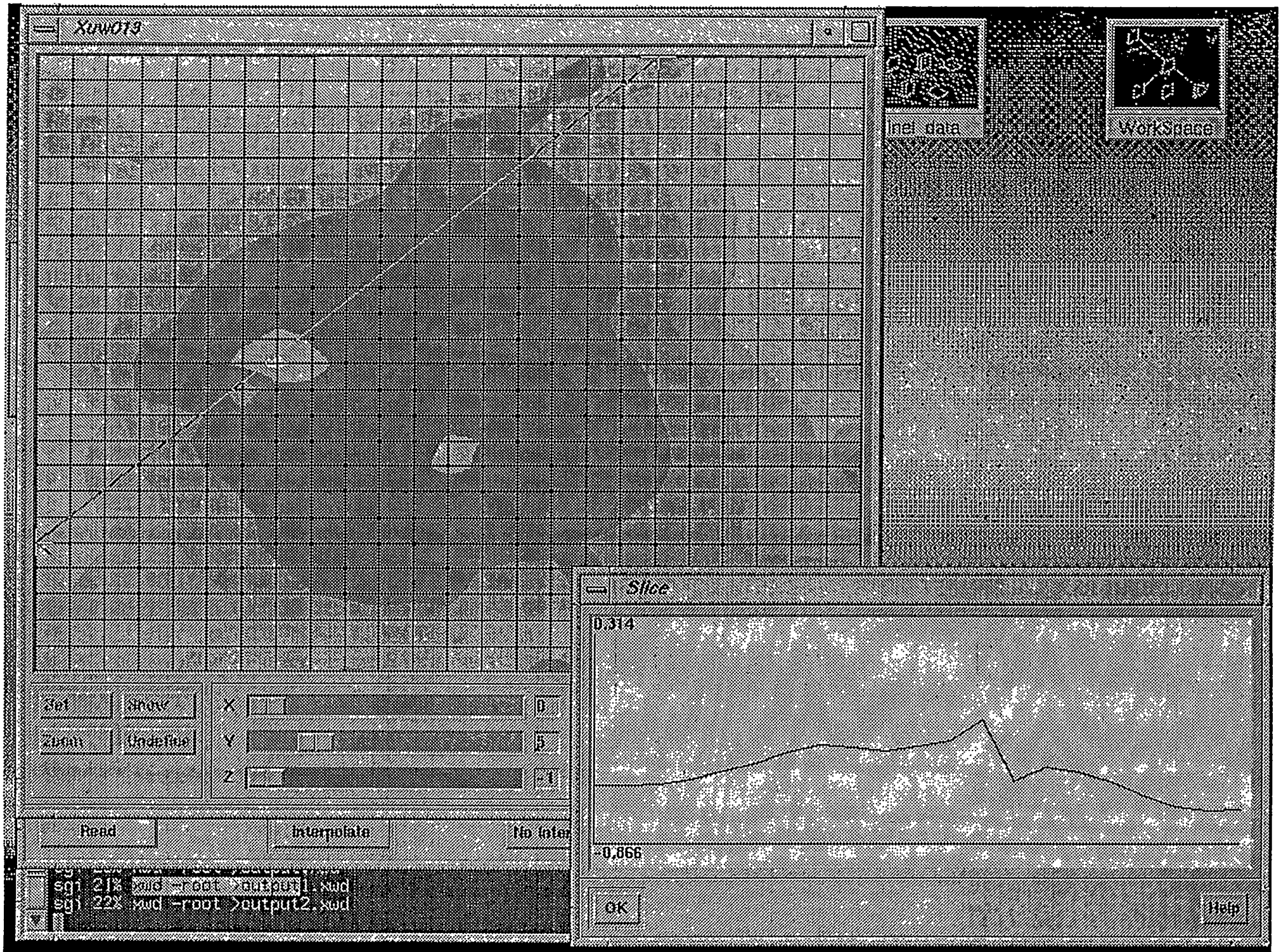
CDisplayOptionsDialog: Class to display a dialog that allows the operator to change any view option for the currently displayed view. It is instantiated when the Display/Options menu button is pressed from the CVdsViewWindow menubar. These options will pertain to only the view in which the display option window was invoked. APPLY, CLOSE, and HELP buttons are supported. Refer to Figure 7-39 for an example of this dialog window.

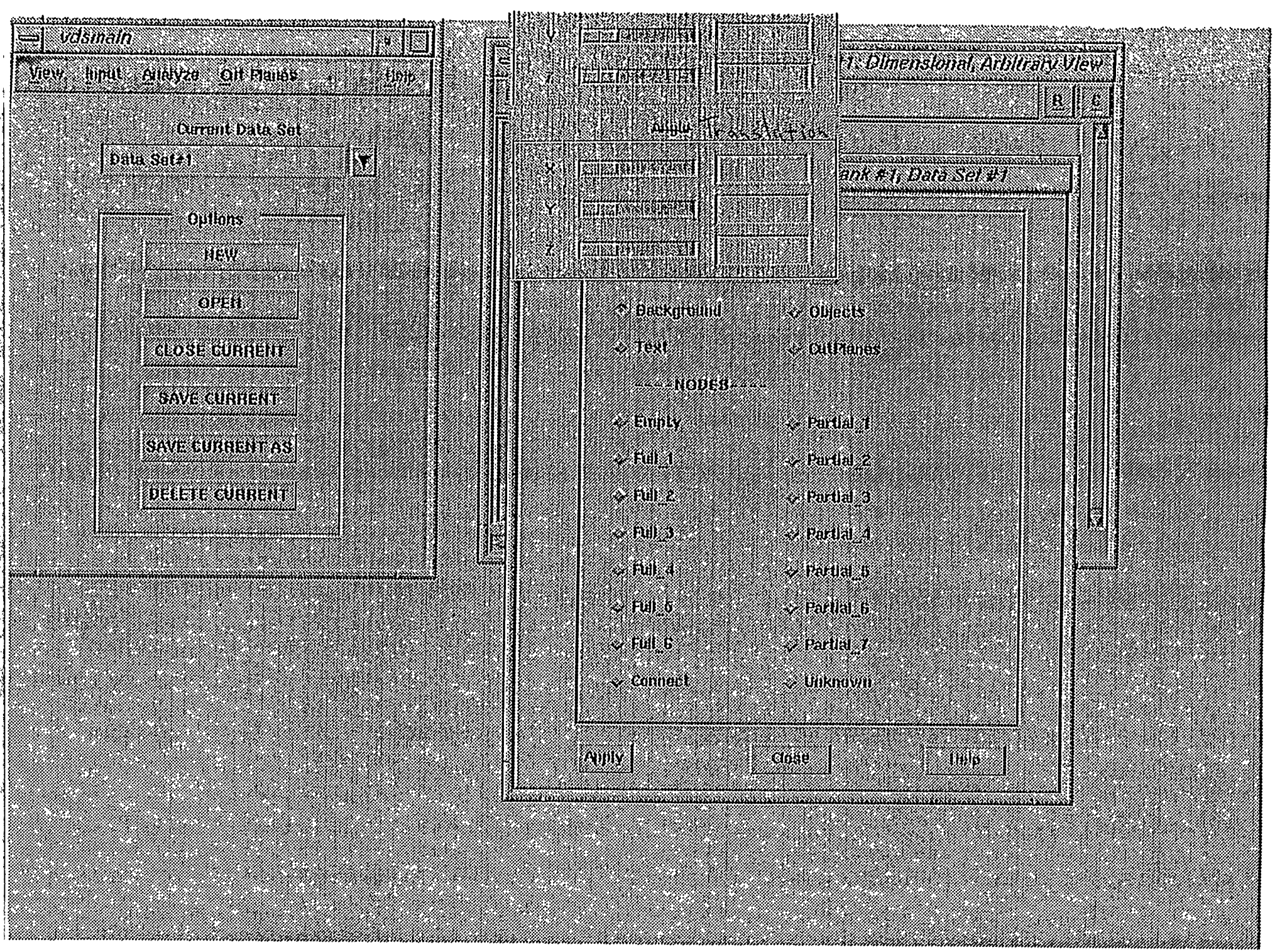
CGeometricObjectFunctionsDialog: Class to display a dialog that allows the operator to perform functions on the geometric objects in the view. This class is instantiated when the OBJECT-Functions menu button is pressed from the CVdsViewWindow menubar. CLOSE and HELP buttons are supported. Refer to Figure 7-20a for an example of this dialog window. This dialog contains 2 subsections:

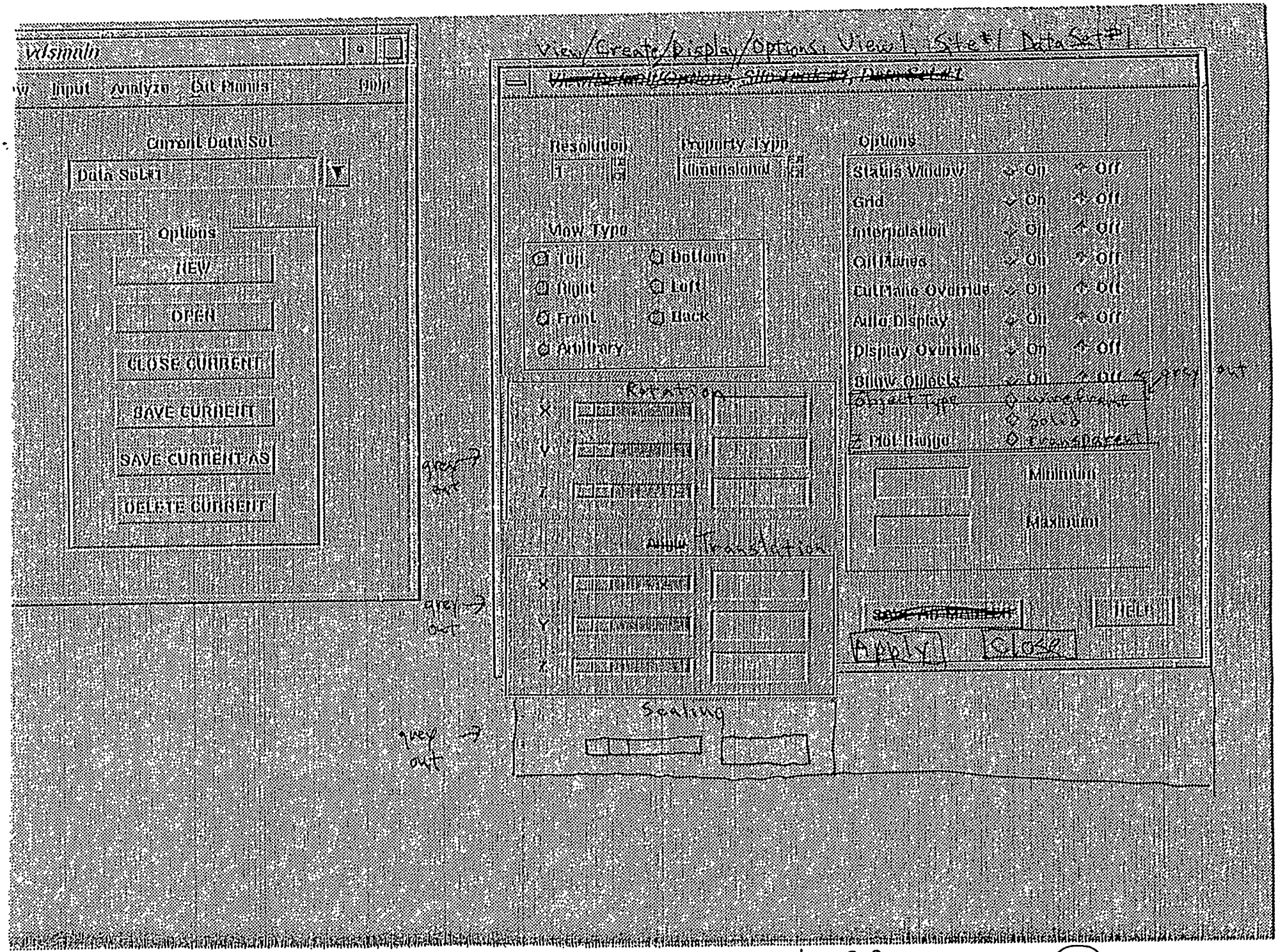
GeometricObjectFunctions: Allows the operator to perform functions on the objects in the view. Object information can be added to the library, deleted, edited, printed, and exported.. In addition a volumetric difference and consistency check can be performd on a selected object or all objects.

GeometricObject Library Functions: Allows the operator to perform functions on the objects in the object library. The object library is a stored set of standard or special geometric objects that may be used when creating a new geometric object.. These objects can be deleted, edited, and printed

CGeometricObjectEditingDialog: Class to display a dialog that allows the operator to interface with geometric objects in the view. The operator can create and edit geometric objects, and save and recall special objects in the object library.. This class is instantiated when the OBJECT-Editing menu button is pressed from the CVdsViewWindow menubar. CLOSE and HELP buttons are supported. Refer to Figure 7-20b for an example of this dialog window. This dialog contains two subsections:


XuwDTG
Inet data
WorkSpace
Slice
0.314
-0,866
Set
Show
Zoom
Undelete
X
Y
Z
0
5
-1
Rend
Interpolate
OK
Help
sgi 21% xwd -root >output1.xwd
sgi 22% xwd -root >output2.xwd


vdsmain
View Input Analyze Off Planes Help
Current Data Set
Data Set#1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
Angle
X
Y
Z
1: Dimensional Arbitrary View
R
C
ank #1, Data Set #1
Background
Text
Objects
OutPlanes
~~~~NODES~~~~
Empty
Full_1
Full_2
Full_3
Full_4
Full_5
Full_6
Connect
Partial_1
Partial_2
Partial_3
Partial_4
Partial_5
Partial_6
Partial_7
Unknown
Apply
Close
Help
~~~~


vdsmain
Input
Analyze
Edit Fields
Help
Current Data Set
Data Set #1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
View/Create/Display/Options: View 1, Set 1 Data Set #1
Resolution
Property Type
Monumental
View Type
Left
Bottom
Right
Left
Front
Back
Arbitrary
Options
Status Window
Grid
Interpolation
Cut Planes
Cut Plane Overlays
Auto Display
Display Overlays
Show Objects
Object Type
Plot Range
Minimum
Maximum
Apply
Close
Help
Scaling

GeometricObjectCreation: Allows the operator to create geometric objects in the view. To create an object, the operator selects a type of object to create and provides the necessary parameters for that selected object type. The operator then presses the Create Prismoid button. The object will appear in the various views for the current dataset. Alternatively, an object can be created from the stored set of library objects. The operator presses the Library Select button, and the object will appear in the various views for the current dataset.

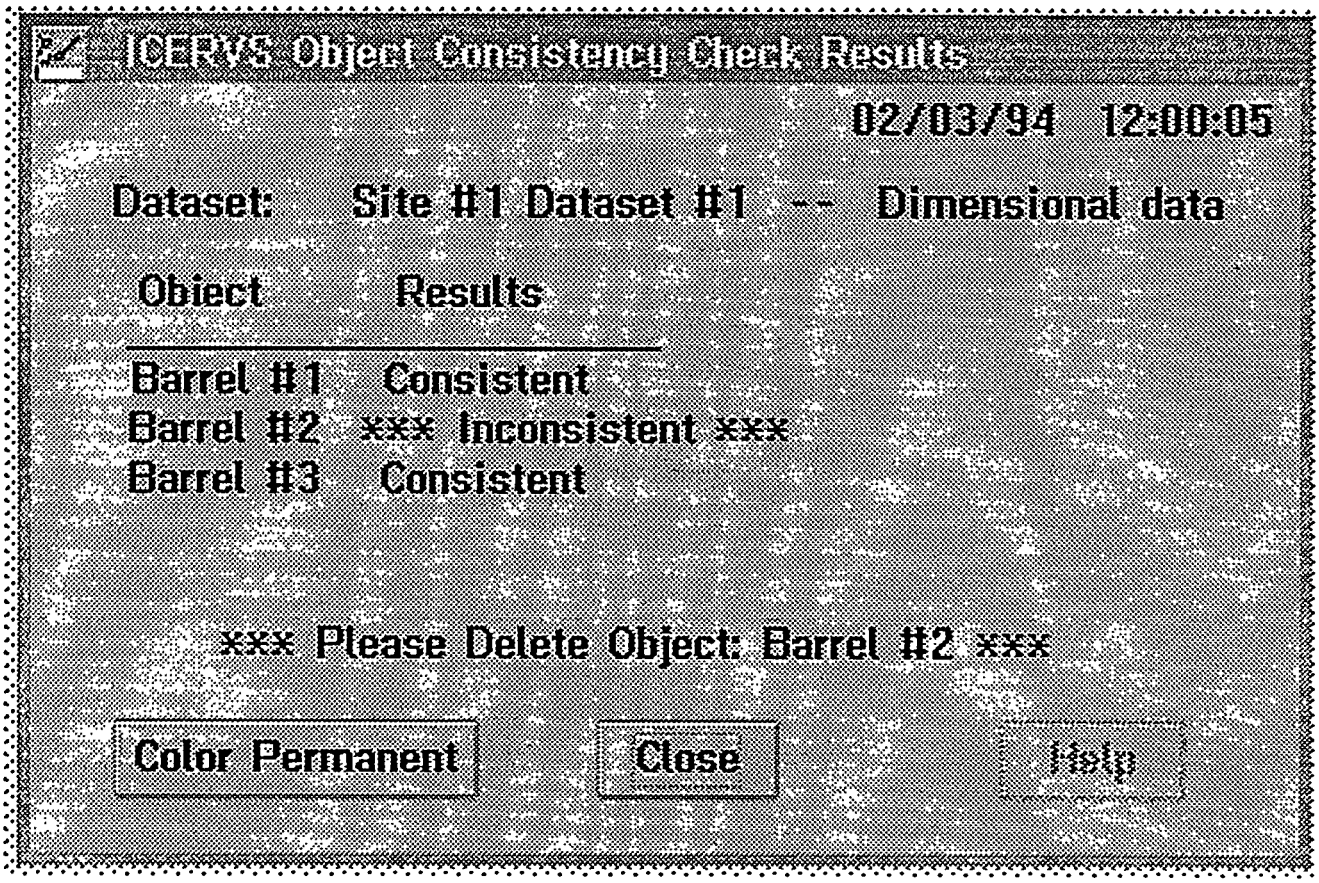
GeometricObjectEditing: Allows the operator to edit objects and/or faces of objects in the view. Editing consists of rotating, translating, and scaling objects or faces of objects, in addition to reshaping faces of objects.

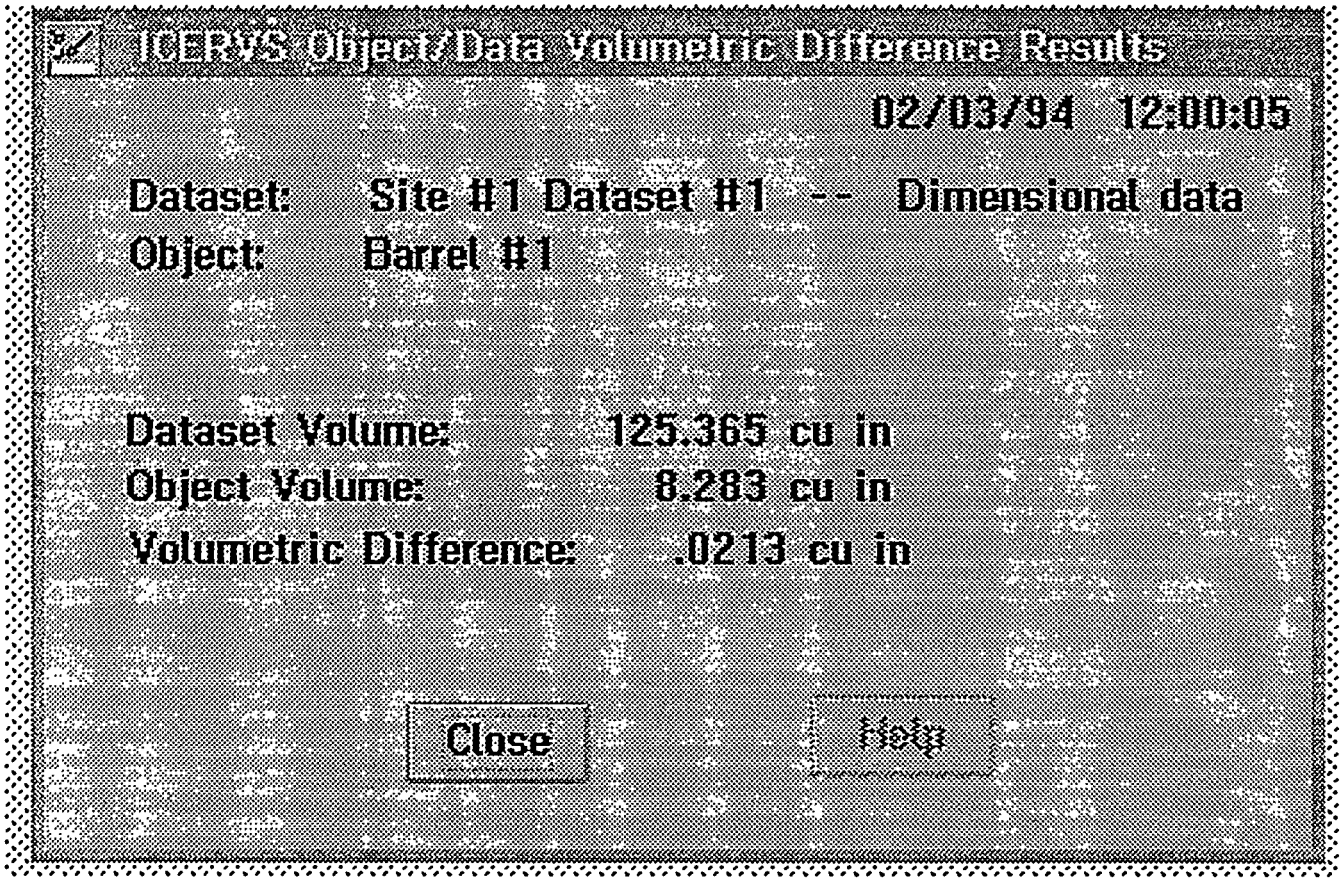
CGeometricObjectEditReshapeDialog: Class to display a dialog that allows the operator to reshape a selected object's faces. Both faces are displayed in this dialog and the operator may select the vertices with the mouse and drag them to a new position. This class is instantiated when the Reshape Faces button is clicked from the GeometricObjectEditing subsection of a CGeometricObjectsEditingDialog. A CLOSE button is supported. Refer to Figure 7-22 for an example of this dialog window.

CGeometricObjectEditTransformDialog: Class to display a that allows the operator to rotate, scale, and translate a selected object or its faces. Sliders are used to reposition the object in the view. This class is instantiated when the Transform button is clicked from the GeometricObjectEditing subsection of a CGeometricObjectsDialog. A CLOSE button is supported. Refer to Figure 7-21 for an example of this dialog window.

CGeometricObjectConsistencyCheckResultsDialog: Class to display the results of a consistency check calculation for a selected or all objects. This class is instantiated when the Consistency Check button is pressed from the GeometricObjectFunctionsDialog. It displays a browse dialog containing the result of the consistency check on the object (consistent or not consistent). Not consistent means that the object is floating in space and should be deleted from the dataset. It also contains a permanent color button to allow the bright color in which any inconsistent objects were redisplayed to become a permanent feature of the object. This bright coloring will ensure that the operator remembers to delete the inconsistent objects at some future point. PRINT, CLOSE, and HELP buttons are also supported. Refer to Figure 7-40 for an example of this dialog window.

CGeometricObjectVolumetricDifferenceResultsDialog: Class to display the results of a volumetric difference calculation for a selected or all objects. This class is instantiated when the Volumetric Difference button is pressed from the GeometricObjectFunctionsDialog. It displays a browse dialog containing the volume of the octree, the volume of the selected object(s), and the volumetric difference between the two. PRINT, CLOSE, and HELP buttons are supported. Refer to Figure 7-41 for an example of this dialog window.


ICERVS Object Consistency Check Results
02/03/94 12:00:05
Dataset: Site #1 Dataset #1 -- Dimensional data
Object Results
Barrel #1 Consistent
Barrel #2 *** Inconsistent ***
Barrel #3 Consistent
*** Please Delete Object: Barrel #2 ***
Color Permanent
Close
Help



Volumetric Difference Results (41)

CRotateRectangleDialog: Class to display a dialog that allows the operator to rotate the data's position within the view. The control rectangle can be rotated to any orientation with the sliders on the attached control panel and the data in the view will be reoriented the same as the rectangle. This dialog is displayed when the "R" button is pressed from the CVdsViewWindow menubar. CLOSE, and HELP buttons are supported. Refer to Figure 7-42 for an example of this dialog window.

CSetofViewWindows: Class that implements a collection of CVdsViewWindow objects.

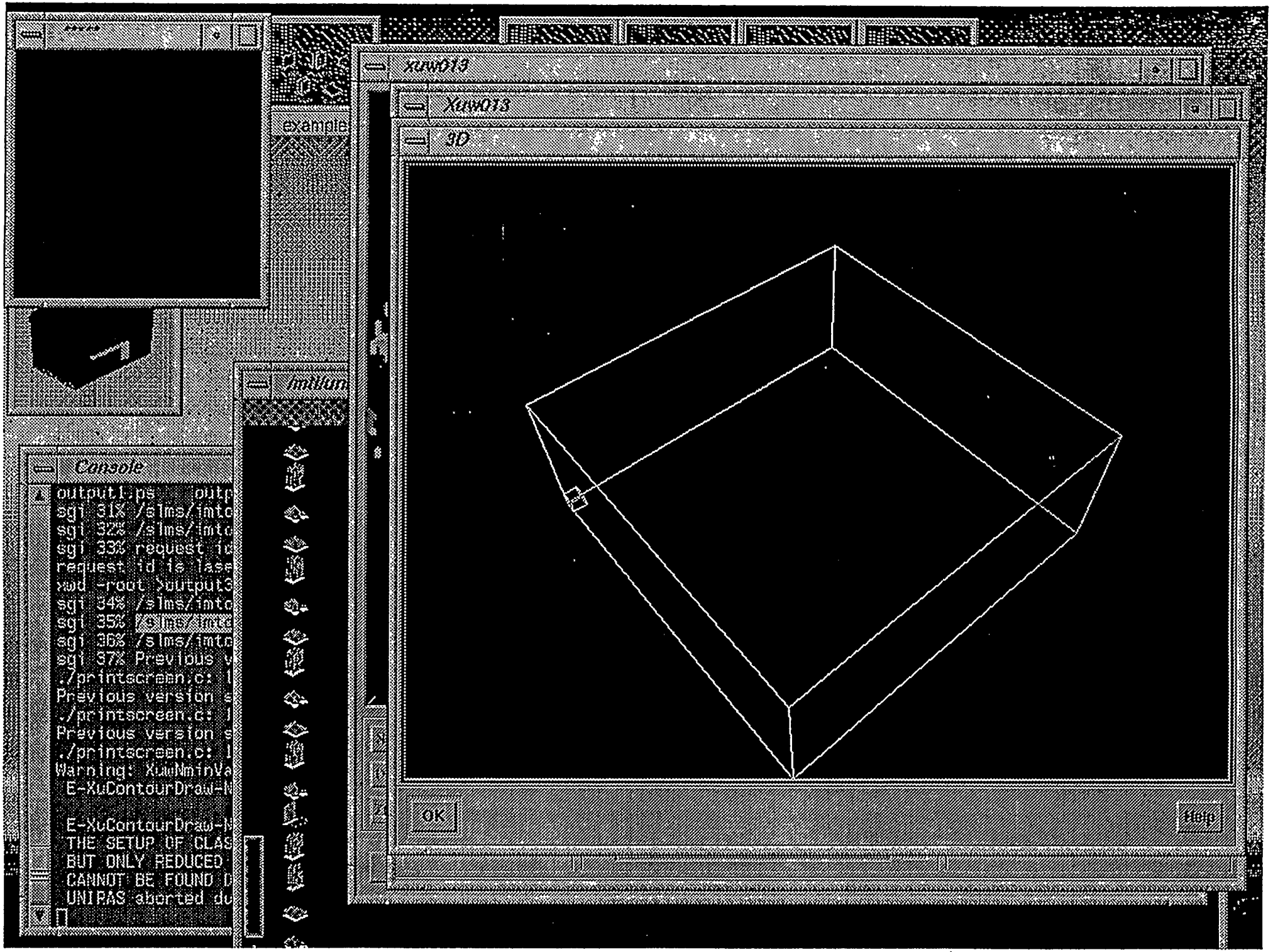
CViewTransformDialog: Class to display a control panel that allows the operator to change the position (for all view types) and rotation (for arbitrary views only) for a view. It is instantiated when the Display/Transform menu button is pressed from the CVdsViewWindow menubar. These transforms will pertain to only the view in which the transform option window was invoked. CLOSE, and HELP buttons are supported. Refer to Figure 7-19 for an example of this dialog window.

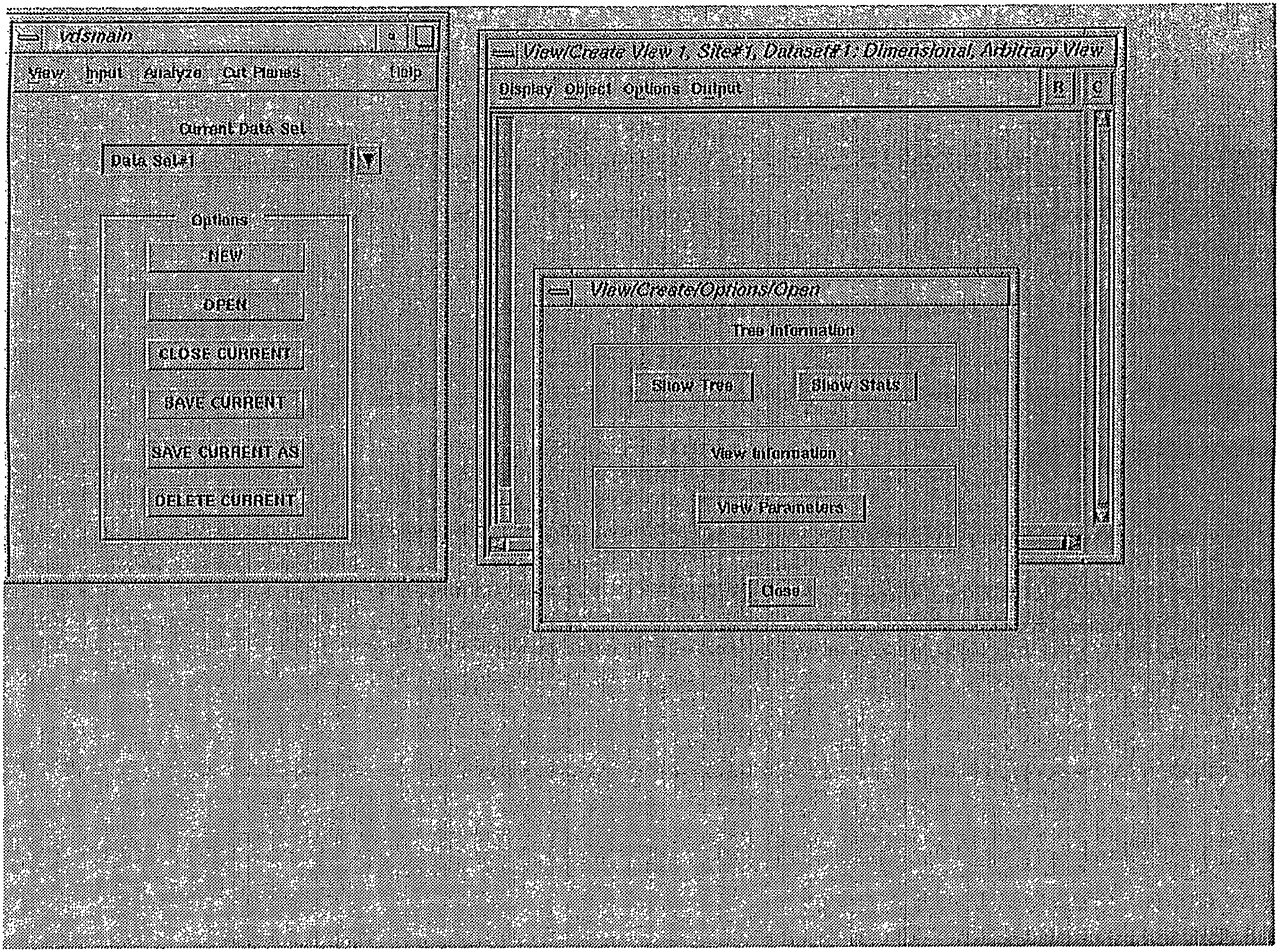
CTreeVolumetricDifferenceWindow: Class to display an analyze/volumetric difference view window. This is actually derived from a CVDSViewWindow with additional things added to make the window into an analyze window. This class is a temporary view window and is not considered part of the set of views for save and restore purposes. It is instantiated when the Compute button on the CTreeVolumetricDifferenceDialog is pressed.

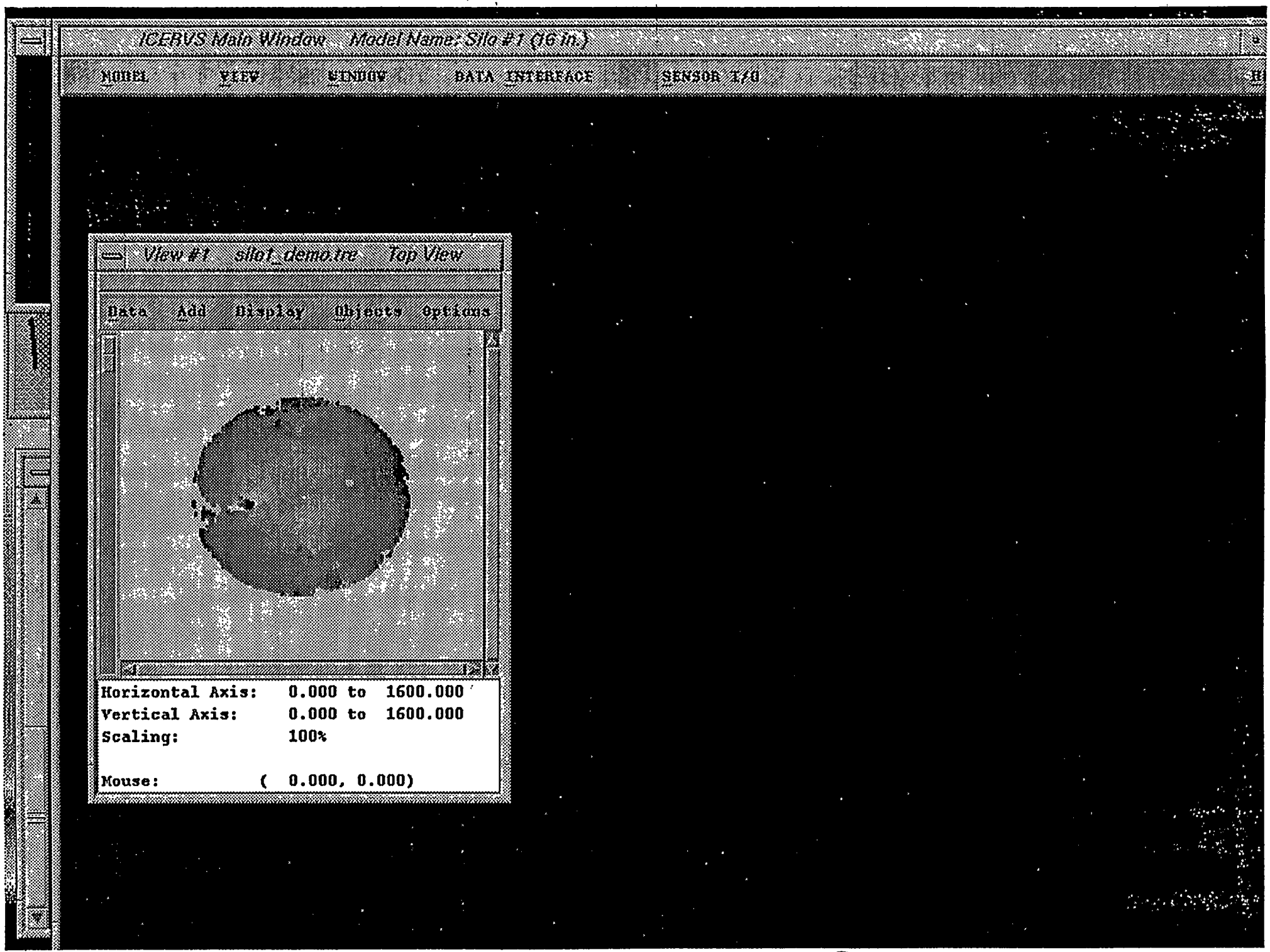
CTree25DSurfaceWindow: Class to display an analyze/2.5D surface view window. This is actually derived from a CVDSViewWindow with additional things added to make the window into an analyze window. This class is a temporary view window and is not considered part of the set of views for save and restore purposes. It is instantiated when one of the Compute buttons on the CTree2.5DSurfaceMapDialog is pressed.

CViewOptionsDialog: Class to display a dialog that contains debugging information for the system. It is instantiated when the Options menu button is pressed from the CVdsViewWindow menubar. CLOSE and HELP buttons are supported. Refer to Figure 7-43 for an example of this dialog window.

CViewStatusWindow: Class to display a status dialog which is attached to the bottom of the view and can be turned on or off at any time. The dialog contains axis information, cutplane information, and the current mouse position. It is instantiated when the status window button is turned ON from the CViewOptionsDialog. No buttons are supported. Refer to Figure 7-44 for an example of this window.


example
XuwDI3
XuwDI3
3D
OK
Help
Console
output1.ps
sgi 31% /slms/imtc
sgi 32% /slms/imtc
sgi 33% request id
request id is lase
xwd -root >output3
sgi 34% /slms/imtc
sgi 35% /slms/imtc
sgi 36% /slms/imtc
sgi 37% Previous v
./printscreen.c: l
Previous version s
./printscreen.c: l
Previous version s
./printscreen.c: l
Warning: XuwNminVa
E-XuContourDraw-N
E-XuContourDraw-N
THE SETUP OF CLAS
BUT ONLY REDUCED
CANNOT BE FOUND D
UNIRAS aborted du


vtkmain
View Input Analyze Cut Planes Help
Current Data Set
Data Set#1
Options
NEW
OPEN
CLOSE CURRENT
SAVE CURRENT
SAVE CURRENT AS
DELETE CURRENT
View/Create View 1, Site#1, Dataset#1: Dimensional, Arbitrary View
Display Object Options Output
R
C
View/Create/Options/Open
Tree Information
Show Tree
Show Stats
View Information
View Parameters
Close


ICERVS Main Window - Model Name: Silo #1 (16 in.)
MODEL
VIEW
WINDOW
DATA INTERFACE
SENSOR I/O
View #1 silo1_demo.tre Top View
Data
Add
Display
Objects
Options
Horizontal Axis: 0.000 to 1600.000
Vertical Axis: 0.000 to 1600.000
Scaling: 100%
Mouse: (0.000, 0.000)

CVdsViewWindow: Class to display a view window and handle all its callbacks. Typically the callbacks make calls to the CVolumetricDataClient class to send commands to the VDS Server program to perform PDC type functions. It is instantiated when the View/Create button is pressed from the CVdsMainWindow menubar. It is derived from a CViewParameters class which obtains its values from the View Parameters File (view.prm). Refer to Figure 7-16 for an example of this window.

CViewParameters: Class to encapsulate the view parameters for a specific dataset. It is a base class for the CVdsViewWindow class and contains the default view parameters file contents initially. It may be subsequently edited for each view window to customize per view.

7.9.4. Major Function Descriptions

The HIC major functions are broken into two groups: the VDS Main Window menu functions and the VDS View Window menu functions. These functions are identified and described below. Their related classes are mentioned where applicable in italics.

7.9.4.1 VDS Main Window Menu Functions

The volumetric data system (VDS) main window *(CVdsMainWindow)* is activated from the Demonstration Application main window's VOLUMETRIC DATA menu (refer to section 6.0) by issuing the VDS_EnableUserInterface command. The VDS main window supports menu functions that pertain to the volumetric data: viewing datasets, inputting datasets, analyzing datasets, creating/deleting datasets, and reading/writing datasets.

A dataset is the main concept in the VDS. A dataset can be thought of as "containing a collection of data that characterizes a waste site at a particular point in time". A list of defined datasets is maintained in a selection list on the CVdsMainWindow. In addition, datasets can be further characterized into groups with attached IDs. Each point in the dataset can have an ID number attached to group the data, if desired. This can be later used to turn on or off groups of data at display time. The operator must select a dataset to work with before performing any functions on the dataset.

The following menu functions are defined for the VDS main window: (Note italicized functions are not implemented in Phase II)

View	Input	Analyze	Cutplanes
Create	Region..Change	Volumetric Difference	Edit
Close All	*Erase*	2.5 D Surface Maps	
Restore	Point.....Erase	*Surface Connectivity*	
Save	Add		
Save All	File.....Add		
Defaults..Options			
Color			

The following button functions are defined for this window: They all have to do with creating, deleting, reading and writing datasets.

New
Open
Close Current
Save Current
Save Current As
Delete Current

VIEW - CREATE *(Create a view of the currently selected dataset)*: Creates a view or set of views for the currently selected data set. Dimensional data is the default property type that is displayed when a view is created. Other property types may be viewed by selecting them from the DISPLAY-Options menu on the CVdsViewWindow display. The type and number of views created is determined by the default view parameters.

VIEW - CLOSE ALL *(Close all view windows for the currently selected dataset)*: Closes all view windows for the currently selected dataset. A dialog to save data *(CCloseViewsVerifyDialog)* before closing the views will be issued if data in any view has changed.

VIEW - RESTORE *(Open a previously saved view)*: Allows selection of a previously saved view file via a CSavedViewFileSelectionListDialog and creates a new view or set of views from the saved view information. View files contain dataset information (i.e. dataset name) as well as view context information (i.e. current display options).

VIEW - SAVE *(Save a selected view in current dataset)*: First selects the CVdsViewWindow to be saved via a list selection box of available views *(CViewSelectionListDialog)*. If only one view, the list box is not necessary. Then brings up a CSavedViewFilesRequestNameDialog for the operator to select the filename for the view to be saved. If the saved file is a new one, the operator must enter the filename and description. View files contain dataset information (i.e. dataset name) as well as view context information (i.e. current display options).

VIEW - SAVE ALL *(Saves all views in the current dataset)*: A verification will be required to ensure that the correct dataset will be used when saving all views *(CSavedViewsVerifyDialog)*. Brings up a CSavedViewFilesRequestNameDialog for the operator to select the filename for the view to be saved. If the saved file is a new one, the operator must enter the filename and description. View files contain dataset information (i.e. dataset name) as well as view context information (i.e. current display options).

VIEW - COPY *(Copy an existing view to a new view)*: First selects the CVdsViewWindow to be copied via a list selection box of available views *(CViewSelectionListDialog)*. If only one view, the list box is not necessary. The view will be copied to a newly created view with all display options duplicated.

VIEW - DEFAULTS *(Brings up the default view editor)*: Either the Default Options Edit window *(CDefaultViewOptionsEditor)* or the Default Colors Edit window *(CDefaultViewColorsEditor)* will be displayed for operator editing. If the operator changes any default settings for the views and saves them as the master file, they will become the new default values. Otherwise the changes are local for the current dataset. Clicking on the OK button will exit the window with editing changes saved. Clicking on the Cancel button will exit the window with no editing changes saved. Refer Figures 7-17 and 7-18 for example screens.

INPUT - REGION - Change *(Change a region)*: Allows the operator to change the node type for a selected region. An Input Region Change window *(CInputRegionsChangeDialog)* is displayed for the operator to define the region to be changed. The region must be defined by entering the correct number of sides for the polygon and pressing the create polygon button. A polygon will appear in the upper right corner of the creation view and can be moved and sized on the view until positioned properly via the Object Edit and Object Create windows. Any other view of the same tree will also show the polygon. The operator can press the Change button to change all nodes in the selected region to the node state entered in the Node Type edit field. To erase a region, simply set the node state to unknown for the selected region. Clicking on the Close button will exit the Input Region Change window. Refer to Figure 7-23 for an example screen.

INPUT - REGION - Erase *(Erase a region in the current dataset)*: Allows the operator to erase the nodes for a selected region. An Input Region Erase window *(CInputRegionsEraseDialog)* is displayed for the operator to define the region to be erased. The region must be defined by entering the correct number of sides for the polygon and pressing the Create Polygon button. A polygon will appear in the upper right corner of the view and can be moved and sized on the view until positioned properly via the Object Edit and Object Create windows. Any other view of the same tree will also show the polygon. The operator clicks the Erase button to change all nodes in the selected region to the UNKNOWN node state. Clicking on the Close button will exit the Input Region Erase window.

INPUT - POINT *(Add or erase a point in the current dataset)*: If erase is the function being performed, the operator must supply the x,y,z, property type, and level (resolution) of the data to be erased *(CInputPointsEraseDialog)*. If "add" is the function being performed, the operator must supply the x,y,z for the spatial information of the point, the property value and type (if any), the resolution (tree level), and the node type *(CInputPointsAddDialog)*. If the inputted property type doesn't exist in the currently selected dataset, a warning should be issued indicating that a new property file is being created for the inputted point. If the property type does exist in the currently selected dataset, then the inputted point is added to the appropriate file in the dataset based on property file. Based on the inputted property type, one of many property value window will pop up allowing a property to be entered. If sculpting is required, then the operator must specify the location and angle of the sensor with which the data was acquired. The newly deleted or added points are updated on the view as well as any other view containing the same data set. Clicking on the Erase or Add button will execute the function. Clicking on the Close button will exit the Input Point Erase or Input Point Add window. Refer to Figure 7-24 for an example screen.

INPUT - FILE *(Add a list of points to the current dataset)*: Allows the operator to add a list of points to the current dataset from an ASCII file. An Input File Add window *(CInputFileDialog)* will be displayed to allow the operator to select some data input options and enter a filename. The file being read will contain data for a given property type which is defined in the file header information. Clicking on the Add button will read in the file. Clicking on the Close button will exit the Input File Add window. Refer to Figure 7-26 for an example screen.

CUTPLANES - EDIT *(Bring up the Cutplane Edit Panel)*: The Cutplane Edit Panel *(CCutplaneEditDialog)* is a modeless window that contains all cutplane parameters for the currently selected dataset. The operator may define up to six cutplanes. Each cutplane is defined by selecting the cutplane to be edited and defining the required parameters for that cutplane. The cutplane is typically a simple plane with a predefined handedness to it (right and below). A half space is created with the side facing to the right and below remaining, and the other half cut away. If the section option is selected for a cutplane, then the cutplane becomes a section plane. This option actually defines a section between two cutplanes spaced a specific width apart (definable by operator). For section planes, it is implied that the sidedness is on the outside of each cutplane such that only the area in between the 2 section planes is visible. When the operator moves a section plane, both section planes move together. Each defined cutplane may be enabled or disabled. When cutplanes are displayed from the view window menubar, only the enabled cutplanes will be displayed. The Cutplane Edit Panel is exited with all editing changes by clicking the OK button. Clicking the Cancel button will exit the window with no editing changes. Refer to 7-29 (or is it 7-33) for an example screen.

ANALYZE - VOLUMETRIC DIFFERENCE *(Compute volumetric difference between two datasets)*: Compute the volumetric difference between two datasets. Computation will include the union, intersection, subtraction, minimum, maximum, and mean of two sets of data. The Analyze Volumetric Difference window *(CTreeVolumetricDifferenceDialog)* is displayed to allow the operator to select the data sets, and the type of function to be performed. When the Compute button is pressed, an Analyze Volumetric Difference view window *(CTreeVolumetricDifferenceWindow)* is displayed with the computed data displayed in it. When the Close button is pressed, the Analyze Volumetric Difference view window is also closed. Refer to Figure 7-27 for an example screen.

ANALYZE - 2.5D SURFACE MAP *(Compute 2.5D surface map for up to two datasets)*: Compute the 2.5D surface map for up two datasets. If one dataset is selected, the 2.5D surface map for only that dataset will be computed. If two datasets are selected, the 2.5D surface map will be the difference between the two. The 2.5D Surface Map window *(CTree25DSurfaceMapDialog)* is displayed to allow the operator to select the data set(s), and the type of function to be performed, along with the data combination type and grid size. When the Compute button is pressed, a 2.5D Surface Map view window *(CTree25DSurfaceMapWindow)* is displayed with the computed data displayed in it. When the Close button is pressed, the 2.5D Surface Map view window is also closed. Refer to Figure 7-28 for an example screen.

NEW *(Open a new dataset)*: Brings up a Dataset Definition dialog *(CNewDatasetDialog)* for the operator to specify the site and description for a new dataset. In addition the operator must specify whether the initial data universe is empty or unknown. The dataset name is generated by the computer (i.e. dataset1, dataset2,...datasetn) and represents the name of the subdirectory within the selected site. The operator clicks on the OK button to continue or the Cancel button to abort the operation, Refer to Figure 7-34 for an example screen.

OPEN *(Open an existing dataset)*:Brings up a dataset selection dialog *(COpenDatasetDialog)* for the operator to select the site and dataset to open. The dataset name contains not only the directory name, but the dataset description and creation date/time. When the site and dataset have been selected, the associated property types are listed for operator viewing. The operator clicks on the OK button to continue or the Cancel button to abort the operation, Refer to Figure 7-35 for an example screen.

CLOSE CURRENT *(Close the current dataset)*: Closes all views for the currently selected dataset. Requires verification if there are any views currently displayed for the selected dataset to be closed *(CCloseViewsVerifyDialog)*...

SAVE CURRENT *(Save the current dataset)*: Saves the current dataset under its existing name.

SAVE CURRENT AS *(Save the current dataset as another dataset)*: Brings up a dialog *(CNewDatsetDialog)* for the operator to define the dataset name for the dataset being saved.

7.9.4.2 VDS View Window Menu Functions

The view windows *(CVdsViewWindow)* are created from the Volumetric Data Window's VIEW-Create menu function. View windows contain information that represents the 3 dimensional waste volume.. They can be either fixed orthogonal or arbitrary views of the data. The type and number of view windows that initially come up are defined in the view defaults file (view.prm). They are Motif windows and contain 2 scroll bars for scrolling or "panning" the image and 1 scroll bar for "scaling" the image. In addition they contain a button on the menu bar labeled "C" for defining and displaying cutplanes. Arbitrary views also contain a button on the menu bar labeled "R" for rotating the image. A view window contains a status window attached to the bottom of it that can be turned on or off. This status window *(CViewStatusWindow)* contains detailed information about the data such as current mouse location, grid size, and cutplane locations. The following menu functions are defined for this window:

Display	OBJECT		OPTIONS	OUTPUT	R	C
Options	Show...	On	Open	Hardcopy		
Color		Off		Postscript		
Transform	Type...	Solid		ASCII file		
Refresh		Wireframe				
ID		Transparent				
	Editing					
	Functions					

DISPLAY - Options *(Bring up the Display Options Control Panel)*: The Display Options Control Panel is a modeless window *(CDisplayOptionsDialog)* that contains all view display parameters. The parameters can be applied to the view by clicking the Apply button. The window can be exited by clicking the Close button. Refer to Figure 7-40 for an example screen. It allows the operator to set values for the following:

Property Type: Sets the property type of the data being displayed. The allowable choices are listed in a pull-down selection dialog. All views default initially to dimensional data and can be changed with this option. The current property type will be highlighted. When the Apply button is clicked, the view will be redisplayed using the new property type.

Plot Range: Sets the plot range for the Z-axis when displaying the data. The minimum and maximum values must be supplied. The current plot range will be displayed. When the Apply button is clicked, the view will be redisplayed using the new plot range.

Tree Level (resolution): Sets the tree level (1-10) to be used for displaying the tree. This is the resolution at which the data will be displayed. The current tree level will be displayed highlighted as the 1st selection of the pop-down select menu. When the Apply button is clicked, the view will be redisplayed using the new level.

View Type: Sets the view type to be used for displaying the tree. The view type can be one of 6 fixed orthogonal viewpoints or an arbitrary viewpoint. The default view type radio button will be highlighted. When the Apply button is clicked, the view will be redisplayed using the new view type.

Status Window: Turns the view window's status window on or off. The status window contains information pertaining to the coordinates (i.e. x,y,z, ranges, cutplane information, current mouse position). The default status window radio button will be highlighted. When the Apply button is clicked, the view will be redisplayed using the new status window option.

Grid: Turns the view window's grid option on or off. The default grid option radio button will be highlighted. When the Apply button is clicked, the view will be redisplayed using the new grid option.

Interpolation: Turns the view window's interpolation option on or off. The interpolation option is used to fill in holes in the data. The default interpolation option radio button will be highlighted. When the Apply button is clicked, the view will be redisplayed using the new interpolation option.

Auto Display: Turns the view window's auto display option on or off. The auto display option is used to turn off automatic redisplay of the data every time that the window is updated.

This can save a lot of time, especially if there are several view on the screen at once that need to be updated. If this option is turned off, the view may be updated manually by clicking on the right mouse button. The default auto display option radio button will be highlighted. When the Apply button is clicked, the view will be redisplayed using the new auto display option.

Display Override: The default display override option radio button will be highlighted. When the Apply button is clicked, the view will be redisplayed using the new display override option.

Cutplane: Turns the view window's defined cutplanes on or off. These cutplanes are defined in the volumetric data window and apply to all views which contain the same data set. The default cutplane option radio button will be highlighted. When the Apply button is clicked, the view will be redisplayed using the new cutplane option. If cutplanes are turned on, then all defined, enabled cutplanes will be displayed on the view. Initially, each cutplane will be located at the position defined in the View-Cutplane-Edit window. Thereafter, cutplanes may be moved by clicking on the "C" on the right part of the View window menubar to enable cutplane movement.

Cutplane Override: Overrides the display of the view window's defined cutplanes. This allows a view to define a cutplane and another view to display the effect of the cutplane.

DISPLAY - Color *(Bring up the Color Control Panel)*: The Color Control Panel *(CDisplayColorsDialog)* is a modeless window that contains all view color parameters. The parameters can be applied to the view by clicking the Apply button. The window can be exited by clicking the Close button. Refer to Figure 7-39 for an example screen. It allows the operator to set values for the following:

Color Options: Allows the operator to set the color of several view features, both general in nature and relating to the node types. The initial color displayed is the default color for the feature which is currently highlighted. The color can be changed by clicking on the Change Color button and selecting the desired color from the Color Selection window *(UNIRAS widget)*. A different feature may be changed by selecting the radio button for that feature and selecting another color. The default color options radio button will be highlighted. When the Apply button is clicked, the view will be redisplayed using the new color options.

DISPLAY - Transform *(Bring up the Transform Control Panel)*: The Transform Control Panel *(CTransformDialog)* is a modeless window that contains all view transformation parameters. The window can be exited by clicking the Close button. Refer to Figure 7-19 for an example screen. It allows the operator to set values for the following:

Scaling: Sets the amount of scaling that the view will perform. The value (between 0-1) can be either set with a slider, input into an edit field, or selected from a list of predefined values. A reset button is available to allow the operator to return to the default value. The default value

will be displayed in the edit field. When changed, the view will be redisplayed using the new scaling parameters.

Rotation: Sets the amount of rotation that the view will perform. The value (between 0-360) can be either set with a slider, input into an edit field, or selected from a list of predefined values. The direction of rotation can be selected by pressing the appropriate button (up, down, left, right, ccw, cw). A reset button is available to allow the operator to return to the default value. The default value will be displayed in the edit field. When changed, the view will be redisplayed using the new rotation parameters.

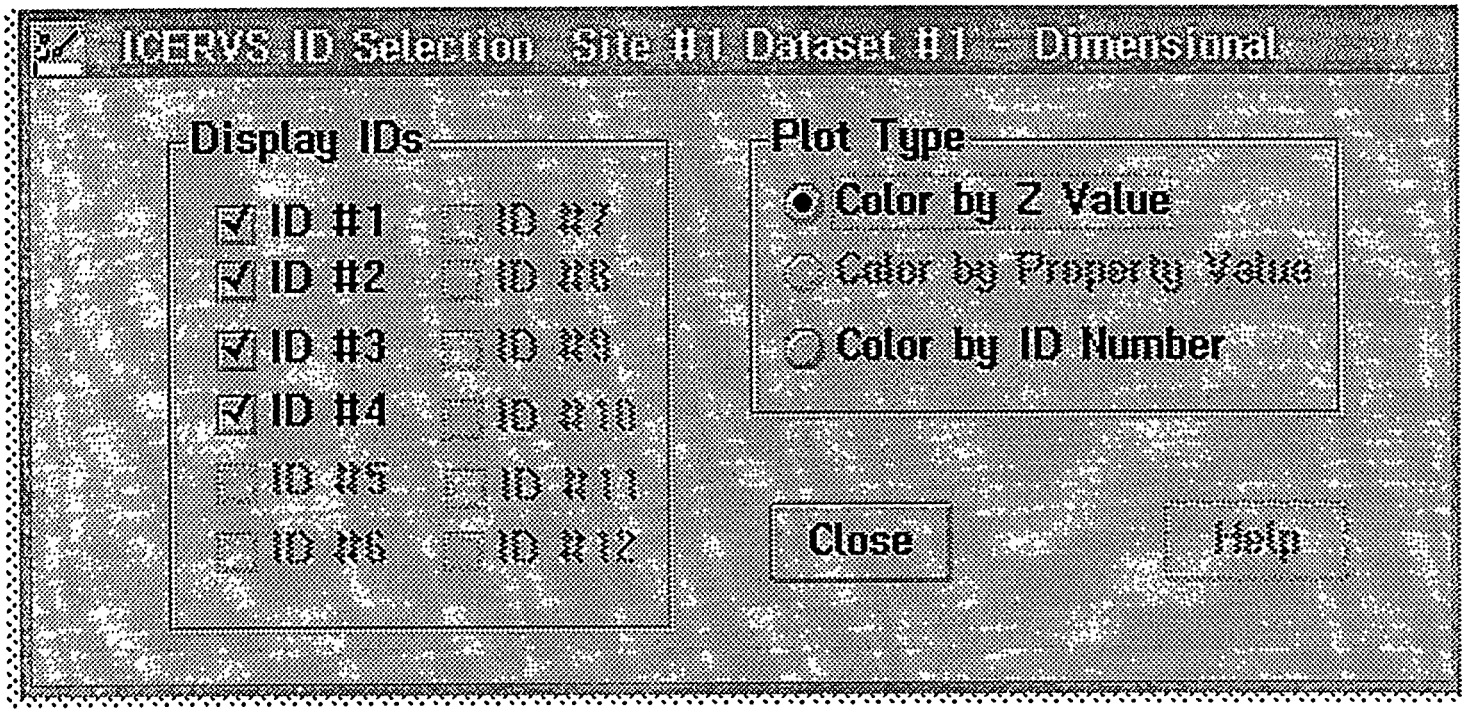
Translation: Sets the amount of translation that the view will perform. The value can be either set with a slider, input into an edit field, or selected from a list of predefined values. The direction of translation can be selected by pressing the appropriate button (up, down, left, right, to, away). A reset button is available to allow the operator to return to the default value. The default value will be displayed in the edit field. When changed, the view will be redisplayed using the new translation parameters.

DISPLAY - Refresh *(Refresh the screen)*: Redraws the screen to update with the latest data and display options.

DISPLAY - ID *(Select the IDs to be displayed)*: Allows the operator to select the IDs to be displayed in the view and the color with which to display each ID.. The dataset being displayed may contain from zero to six different IDs. That is, each data point in the dataset is tagged with an ID between 0 and 12. These IDs correspond directly to the node type in the tree. The IDs contained in the dataset are listed in a multiple selection window for the operator to select which ID(s) will be displayed and how to display them. Each ID may be displayed colored by Z value or property value (depending on what property type is being displayed), or they may be displayed colored by ID or node type (the color for each node type can be set in the DISPLAY COLOR option). Initially, all IDs are displayed colored by Z value or property value (depending on what property type is being display). This is a way for the operator to display multiple dig sites of data on the same view for comparison purposes. See Figure 7-45 for an example of this screen.

OBJECT - Show *(Turn ON or OFF the object in the view)*: This option allows the geometric objects to be displayed or hidden in a view. None of the other object functions can be selected until this option has been turned ON.

OBJECT - Type *(Select the object display type)*: This option allows the geometric objects to be displayed one of three different ways (solid, wire frame, or transparent). This option does not apply when the objects are being edited. During editing, the objects are always wire frame. This is disabled until the object option Show Object is turned ON.



Selection ID 45

OBJECT - Editing *(Bring up the Object Editing Dialog)*: The Object Editing Dialog *(CObjModelEditingIDialog)* is a modeless window that contains all object editing parameters. These are the parameters needed to create and edit volumetric objects. Two categories of parameters are displayed: Object Creation, and Object Editing. The window can be exited by clicking the Close button. This is disabled until the object option Show Object is turned ON. Refer to Figure 7-20b for an example screen. It allows the operator to do the following:

Object Creation: Allows the operator to create an object on the view by either creating a prismoid from scratch, or selecting a predefined geometric object from the object library. If creating an object from scratch, the operator must enter the dimensions of the object (number of vertices, radius and length) and then press the CREATE PRISMOID button to create the object. If selecting an object from the object library, the operator must press the LIBRARY SELECT button to select the object. Either way, the object will appear on the view. If the view is an arbitrary view, the object will be 3 dimensional otherwise the object will be 2 dimensional. The operator will be automatically prompted to identify the object being created with a name. Any other view of the of the same tree which has objects enabled will also show the object.

Object Editing: This object category is used to change the shape or position of the currently selected object. An object must be selected first with the mouse button or via the object selection list in order to edit the object. When the Transform button is pressed, the Object Editing Transform Window *(CObjModelTransformDialog)* is displayed and the operator is given the ability to set the scale, rotate, and translate parameters to change the placement of the selected object or faces of the object. When the Reshape Faces button is pressed, the Object Editing Reshape Faces Window *(CObjModelReshapeDialog)* is displayed, showing the two end faces of the object and allowing the operator to change the shape of one or both faces.

OBJECT - Functions *(Bring up the Object Functions Dialog)*: The Object Functions Dialog *(CObjModelFunctionsDialog)* is a modeless window that contains all object functions. Two categories of functions are displayed: Object Functions and Library Functions. The window can be exited by clicking the Close button. This is disabled until the object option Show Object is turned ON. Refer to Figure 7-20a for an example screen. It allows the operator to do the following:

Object Functions: This object category is used to execute the object functions. Each function applies to the currently selected object.

Library Add (Add an object to object library) Allow the operator to add an object or all objects in the view to the object library. The object library is a storage location for special or standard geometric objects that may be used for subsequent object creation. The object must have been first selected via the object selection list to identify which object is to be added to the library, otherwise the menu item is not active.

Delete (Delete a volumetric object): Allow the operator to delete an object or all objects from the view. The object must have been first selected via the object selection list to identify which object is to be deleted, otherwise the menu item is not active.

Edit (Modify object information): Allow the operator to edit an object or all object's information. The object information includes object name, color, description, type, and data vertices. The object must have been selected via the selection list to identify which object's information is to be viewed, otherwise the menu item is not active.

Print (Print object information): Allow the operator to view and print an object or all object's information via a scrollable browse window. The object information includes object name, color, description, type, and data vertices. The object must have been first selected via the object selection list to identify which object's information is to be printed, otherwise the menu item is not active.

Information (Get short information on the selected object): When this button is clicked on, some identification of the object will show up about the currently selected object such as its name and volume, etc.

Export (Export object information): Allow the operator to export an object or all object's information to a file of another format. The object must have been first selected via the object selection list to identify which object's information is to be exported, otherwise the menu item is not active.

Volumetric Difference (Compute volumetric difference): Allow the operator to check an object or all objects to compute the volumetric difference between an octree solid model and the object model. The object must have been first selected via the object selection list to identify which object's information is to be computed, otherwise the menu item is not active. The results are display in the Object Volumetric Difference Dialog window *(CObjVolumetricDifDialog)*.

Consistency Check (Scan object for consistency (suspended in air): Allow the operator to check an object or all objects to determine whether the object(s) are suspended in air. The object must have been first selected via the object selection list to identify which object's information is to be computed, otherwise the menu item is not active. The results are displayed in the Object Volumetric Difference dialog window *(CObjConsistencyCheckDialog)*.

Library Functions: This object category is used to execute the library functions. Each function applies to the currently selected object.

Delete (Delete a library object): Allow the operator to delete an object or all objects from the object library. The object must have been first selected via the object library selection list to identify which object is to be deleted, otherwise the menu item is not active.

Edit (Modify library object information): Allow the operator to edit a library object or all library object's information. The object information includes object name, color, description, type, and data vertices. The object must have been selected via the object library selection list to identify which library object's information is to be viewed, otherwise the menu item is not active.

Print (Print library object information): Allow the operator to view and print a library object or all library object's information via a scrollable browse window. The object information includes object name, color, description, type, and data vertices. The object must have been first selected via the object library selection list to identify which library object's information is to be printed, otherwise the menu item is not active.

Information (Get short information on the selected library object): When this button is clicked on, some identification of the object will show up about the currently selected library object such as its name and volume, etc.

OPTIONS - Open *(Bring up the Options Control Panel)*: The Options Control Panel *(CViewOptionsDialog)* is a modeless window that contains all options parameters. These are typically options that are used for debug purposes. The window can be exited by clicking the Close button. Refer to Figure 7-44 for an example screen. It allows the operator to set values for the following:

Show Tree (*Show the tree nodes*): Shows the tree node information in a scrollable file for operator browsing. Used for debug only.

Show Stats (*Show the tree statistics*): Shows the tree statistics information in a scrollable file for operator browsing. Used for debug only.

View Parameters (*Show the view parameters)*: Shows the view parameters in a scrollable file for operator browsing. Used for debug only.

OUTPUT - HARDCOPY *(Print the screen)*: Print the screen to a connected printed. If no printer exists, the screen will be dumped to a Postscript file named: "screendump.ps".

OUTPUT - POSTSCRIPT (*Output the screen to a Postscript file*): Outputs the screen to a Postscript file. A dialog (CNewFileSelectionDialog) is display to get the name of the Postscript file to be generated..

OUTPUT - ASCII FILE *(Output the data in the view to an ASCII file)*: Outputs the data in the view (x,y,z,property value) to an ASCII file. A dialog (CNewFileSelectionDialog) is displayed to get the name of the ASCII file to be generated. This file can later be used for post processing with another software package.

C *(Redefine Cutplanes)*: The Cutplane Selection Control Panel *(CCutplaneSelectDialog)* allows the operator to redefine the cutplanes. Refer to Figure 7.9.1.6-1 for an example of this dialog window. This can be done by selecting a cutplane and editing its values. Four types of cutting functions can be generated.

1. A cutplane is a defined plane that cuts the data in the view. All data defined to the right and below the cutplane will be clipped from the view. A cutplane cube window allows the operator to redefine a cutplane and then apply it to the current dataset. Refer to Figure 7-30 for an example of this window. This is done by selecting a cutplane from a selection list of all cutplane definitions on the Cutplane Selection Control Panel and clicking on the Cutplane button. When the cutplane is selected, it should become highlighted and ready for motion. A cutplane cube and translation box will be displayed in a Cutplane Redefinition window *(CCutplaneCubeDialog)* for the operator to use for translation and rotation of the selected cutplane. Moving the cube with the mouse will rotate the cutplane accordingly. Moving the sliders in the translation box will translate the cutplane accordingly. Once the cutplane has been redefined in the view, all other linked views containing the same data which have cutplanes enabled will update their screens to display the data with the latest cutplane definition.

2. A section is a single cutplane which has some width associated with it. Only the data contained in the section will be displayed. This section can then be selected and dragged to the desired location and the view will be updated to reflect the latest position. A section plane cube window allows the operator to redefine a cutplane and then apply it to the current view. Refer to Figure 7-30 for an example of this dialog window. This is done by selecting a cutplane from a selection list of all cutplane definitions. on the Cutplane Selection Control Panel and clicking on the Section button. A section plane cube and translation box will be displayed in a Cutplane Redefinition window *(CCutplaneCubeDialog)* for the operator to use for translation and rotation of the selected section plane. It is here that the width of the section plane can be defined. Moving the cube with the mouse will rotate the section plane accordingly. Moving the sliders in the translation box will translate the section plane accordingly. Once the section plane has been redefined in the view, all other linked views containing the same data which have cutplanes enabled will update their screens to display the data with the latest section plane definition.

3. A profile is a 2 dimensional, horizontal or vertical slice of the data for the current view. A profile window *(UNIRAS ContourXplore)* allows the operator to view a profile of the data for either a horizontal or vertical position.. This is done by clicking on the Profile button from

the Cutplane Selection Control Panel. Any cutplane selection is ignored here since the profile function has nothing to do with the predefined cutplanes. Moving the sliders in the profile window will display the profile of the data at the point indicated by the line on the data. No views will be updated as a result of this function

4. A slice is a 2 dimensional, arbitrary slice of the data for the current view. A slice window *(UNIRAS ContourXplore)* allows the operator to view a slice of the data for any arbitrary position.. This is done by clicking on the Slice button from the Cutplane Selection Control Panel. Any cutplane selection is ignored here, since the slice function has nothing to do with the predefined cutplanes. The operator must define 2 endpoints of a line and then move that line with the mouse to display the slice of the data at the point indicated by the line on the data. No views will be updated as a result of this function

R *(Rotate data in view)*: The rectangle rotation box *(CRotateRectangleDialog)* allows the operator to rotate the data in the view. This can be done by orienting the rectangular box with the mouse to the same position that the data should be positioned. This window can be exited by clicking the CLOSE button.