# X-Core and Viz:
# Managing Data from Ships and Robotic Vehicles

## DRAFT DRAFT DRAFT

**Schwehr K[1], Nguyen L[2], Derbes A[3], Edwards L[4], Zbinden E[4]**

1. Scripps Institution of Oceanography
2. Intel
3. GGHC Skunkworks
4. NASA Ames Research Center

## 0. ABSTRACT

One of the challenges prevalent in visualization software design is how to provide tools capable of concurrently displaying data that varies in scale from kilometers to micrometers, such as the data prevalent in planetary exploration and deep-sea marine research. The Viz software developed by NASA Ames and the more recent X-Core extensions provide a flexible framework for rapidly developing visualization software capable of accessing and displaying large data sets. This paper describes the Viz/X-Core design and illustrates the operation of the Viz/X-Core system over a number of deployments ranging from marine research to Martian exploration. Highlights include a 2002 integration with live ship operations and the planned Viz/X-Core implementation for the 2004 Mars Exploration Rovers.

**KEYWORDS:** OpenInventor, Robotics, Multiple Views, Scene Graph, Distributed Visualization, Geology

### 1. INTRODUCTION

For the last 15 years, NASA Ames has been developing virtual environment software for a wide range of vehicles, including spacecraft and underwater vehicles. The first work at NASA Ames on vehicle interfaces was the Virtual Interface Environment Workstation (VIEW) (Fisher, 1991). This project focused on human-computer interaction with CAD models and simple indoor robots. In 1992, development began on underwater Remotely Operated Vehicle (ROV) control systems, using the Telepresent ROV (TROV). The TROV has been deployed to sites ranging from Lake Tahoe to the dry valleys of Antarctica (Figure 1.1). Subsequent to VIEW, the Virtual Environment for Vehicle Interface (VEVI) control software was developed for the second TROV Antarctic deployment in McMurdo Sound (Hine et al., 1994). VEVI was used on a number of projects including the Ranger spacecraft and the NASA Ames Marsokhod (Figure 1.2). VEVI was initially implemented with Sense 8 WorldToolKit, RTI Control Shell, and RTI NDDS (Pardo-Castellote and Schneider, 1994). After release 4.0 in 1996, VEVI was adapted by Fourth Planet to use their own proprietary networking package and sold in a range of visualization products.

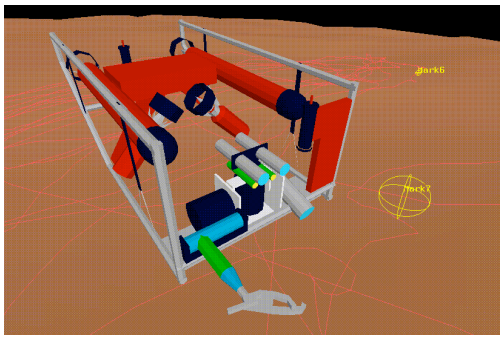

**Figure 1.1**: TROV viewed in VEVI while working in McMurdo Sound. Shown are a live ROV model, 3D vehicle track, and station markers.

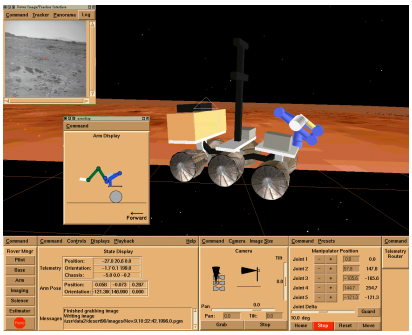

**Figure 1.2**: VEVI controlling the NASA Ames Marsokhod in the Arizona desert.

Following VEVI, NASA Ames developed MarsMap for the Mars Pathfinder mission in 1996 (Stoker et al., 1999). It utilizes a design incorporating all functionality into a single monolithic program. Unfortunately, MarsMap instances running on different hosts cannot interact with each other and all data input and output requires user intervention. However, the scientists in the Pathfinder science operations working group clearly liked the capabilities of MarsMap and requested additional functionality and the ability to use the software at their home institutions.

The 1997 Carnegie Mellon/NASA Ames Atacama Desert Trek used a novel interface for streaming video back from the Nomad rover. The Telepresent Interface (Thomas, 2003) took images from an upward looking camera pointed at a spherical mirror. The data was then multicast via NDDS over a T1 satellite link from the Chilean high desert back to the United States to multiple workstations in Mountain View, California and Pittsburgh, Pennsylvania. The interface dewarped each image and displayed it on a spherical mesh in a custom SGI Performer application. Each workstation had its own separately controlled point of view into the full 360 scene using either the mouse or an attached Magellan Space Mouse (3Dconnexion Inc.) control. Independent viewpoints were available for each science or engineering team member turned which out to be extremely important to the efficiency of the teams. The distributed networking concept used for the Telepresent Interface was the basis for the distributed control and shared data viewing capabilities incorporated in Viz.

Based on scientists experience with MarsMap, the Nomad Telepresent Interface, and their feedback, NASA Ames developed Viz using OpenInventor (Wernecke, 1994) for deployment during the 1998 Mars Polar Lander Mission (Figure 1.3). Viz implements an extensible networked architecture and was designed to overcome some of the limitations of the MarsMap approach and incorporate lessons learned from previous mission experience. In 2001, after the initial development and deployment of Viz, several opportunities for extending Viz presented themselves in the area of marine geological research, which were implemented as the X-Core extension. X-Core and Viz is described in detail below.

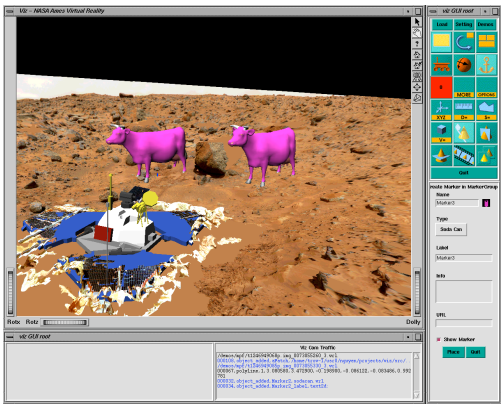

**Figure 1.3**: The Viz architecture applied to a Mars Pathfinder data set. Cows are present as a scale reference.

In late 1997 Fourth Planet, in cooperation with NASA Ames, produced a browser-based system using VRML and Java to try to provide a universally available visualization and control platform. Vermilion used the Cosmo plugin (then owned by SGI) with Netscape Navigator using the EAI programming interface. While the project succeeded, it clearly illustrated the instability of the EAI interface and the Cosmo plugin. Unfortunately, the Cosmo plugin has not been updated since 1998 and is restricted to Microsoft Windows and the Netscape 4.0x browsers.

Since the development of Vermilion, the GeoVRML working group of the Web3D consortium has created the GeoVRML system for dealing with geographic datasets (McCann, 2002; Reddy et al., 2000). GeoVRML is an additional set of nodes on top of VRML97 and requires CosmoPlayer or Cortona plugin. MBARI currently uses GeoVRML for visualization of ROV and AUV operations. We prefer to avoid browser centric solutions, but with the continual improvements with OpenInventor libraries, GeoVRML may soon be usable with OpenInventor.

There have been a number of commercial applications in the area of marine visualization that have approached similar problems to spacecraft. IVS has built the Fledermaus (Fonseca et al., 2000), which focuses on the oil and gas industry. Fledermaus can import of a wide range of data formats, has extensive color mapping tools, and provides basic spatial query tools. For example, it is able to draw a line in the world and get back the topography for that cross section. Recently, Fledermaus was extended to include ArcView based database access tools. This work integrates a wide range of datasets from the STRATAFORM project field site off of the Eel River in northern California. These tools allow the user to view the locations of cores in ArcView and select cores to pop up 2-D graphs of core logs.

Finally there is another underwater vehicle visualization system called GeoZui3D (Ware et al., 2001), which was designed by the University of New Hampshire as an extension to the Fledermaus capability. It supports the ability to pipe in real-time data via a CORBA interface. It provides basic split-screen viewing and basic ROV/AUV tracking.

## 2. METHODS

### 2.1 The Viz Rendering Server

The challenge behind the design of Viz architecture was to create a system that is generic and only deals with geometric entities whose characteristics can be changed on the fly (color, positions, etc). The system should be robust enough to render the scene at a reasonable frame rate under various constraints (geometries with high number of polygons, high update rate, multiple data inputs) and also be flexible enough to make the integration easier by the application developer.

Viz/X-Core is designed around a central rendering server on each workstation (Figure 2.1.1). This server can spawn subprograms such as OpenInventor's ivview and xcore-ivview, Netscape and Mozilla browsers, shell commands, and speech synthesis programs. The Viz server, known as "mainViz," starts up and connects to a pre-defined port number on a TCP socket. Then any number of client programs can connect to the server and alter the scene graph contained in the mainViz program.

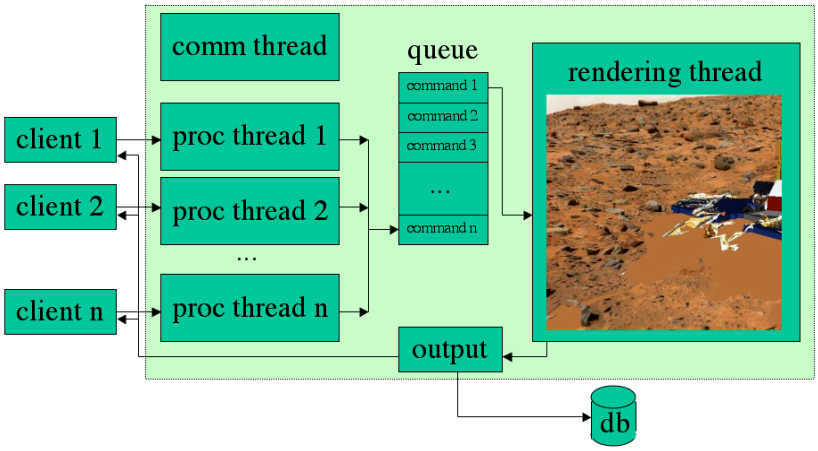


**Figure 2.1.1**: Overview of the Viz rendering architecture.

Client programs can be of two basic types. The simplest class of clients falls into the group of automated programs and shell scripts. These clients can be used to do such tasks as adding new models as they are available or updating nodes with new value. The second class of clients is user interface agent. In the traditional monolithic design, the User Interface (UI) is directly embedded in the server. We have left the basic navigation via mouse or Magellan Space Mouse in the standard OpenInventor Examiner Viewer mode. All other user interfaces are implemented externally. This allows user interfaces to be written using the toolkit of the developer's choice. For example, with the Mars Polar Lander, the UI was developed in Python/Tk while other interfaces can be designed in Cocoa, Qt, X11, etc. It is easy to create a mixed UI with parts using different UI toolkits to leverage the strengths of different toolkits and/or developer knowledge and experience.

Another major component of power and usability of the Viz render server comes directly from using OpenInventor. Using manipulators, draggers, calculators, and engine nodes in an ASCII OpenInventor file, the system is able to import functionality with the scene graph without any modifications to the server.

With this kind of design, it is even possible to create a rendering server in a different graphics APIs such as IRIS Performer or WorldToolKit without loosing the overall design and infrastructure. You would, however, loose the many advantages that come from OpenInventor, such as draggers and manipulators.

### 2.1.1 Network model

We implemented Viz as a server-client model. The server takes care of rendering the scene, while listening to messages sent by clients. Clients are typically application-specific and developed by the user of Viz. For instance, in order to display live telemetry updates, the client receives telemetry updates and sends appropriate messages to the server such as: change color of motors (display temperature), change position/orientation of robot, add obstacle as new object in 3D scene.

The advantage of this approach is that multiple clients can be connected to the server (e.g. one for each source of data) and can independently send their updates. Similarly, some of our applications require being able to run multiple Viz servers that render the same scene on machines located in different institutions, with distributed science operations. One client can connect to several different servers and update them all simultaneously. Moreover, we drastically reduced the amount of effort in the client development by

reducing the graphical operations into a few set of commands wrapped into a Viz API that are then used by the client.

### 2.1.2 Message encoding

The message system is built on the eXternal Data Representation (XDR) as specified in Internet RFC 1832 (Srinivasan, 1995). We chose to utilize XDR based on a number of factors. Most importantly, XDR encoding/decoding is available in standard C libraries for just about every platform and is also available for a wide range of programming languages. XDR is a relatively simple specification that includes a basic Yacc/Bison grammar as a part of the specification. Therefore we were able to quickly build a code generator that produces ACSII, C, Python, Java, and Lisp encode and decode routines for each message type.

There are a number of alternative messaging/middle-ware programs (including NDDS and CORBA) but we found that they were too large and complex for the Viz design. We wanted to be able to send messages from simple shell scripts, embedded controllers, and in a range of languages. The Viz server and clients communicate with XDR directly over TCP/IP.

There are currently 39 messages in the Viz 1.0 implementation. These commands are arranged into 4 groups: render core, administrative, mode, and query. The render core commands provide basic scene graph manipulation, which include adding and deleting objects, changing their color, scale, position and orientation, and hiding and showing an object. The administrative commands control the viewpoints, cursor, and screen geometry, save scene graphs to disk, and control timers and switches. The 3 mode commands select between normal, polyline generation, and selection modes for mouse interaction in the viewer window. Lastly the query mode lets any client ask the render server for the state of polyline generation, pose, and ray picks, get a back an elevation grid for a user selected area, and discover the names of objects in a sub-scene graph.

### 2.1.3 Queuing, locking, and pthreads

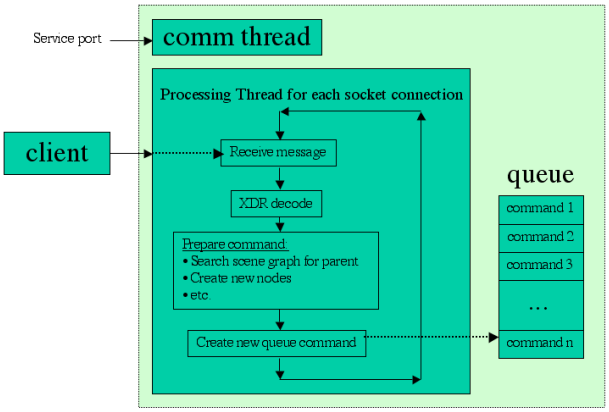


**Figure 2.1.3.1**: Per-thread communication model for queuing commands.

In order to avoid blocking on commands between clients, the render server and disk access, we implemented a threaded render server using the pthreads library with a queue into which all commands are entered. There is a timer on the render server (defaulting to every 20 milliseconds) that, when triggered, executes all the commands cached in the queue. Each client that connects to the render server is allocated a thread that manages the connection and tries to do as much preprocessing before entering the command into the server's queue (Figure 2.1.3.1). We did a rough analysis of thread safety in the SGI OpenInventor library and tried to design a system that could do as much as possible while not stepping on the OpenInventor rendering thread. Commands are able to search the scene graph for the named node or load data from disk and build scene graphs within this thread. This threaded distribution of processing takes advantage of multiple CPU's when available and keeps rendering delays to a minimum for updates. A

semaphore protects command insertion into the queue and the processing of commands by the render thread.

This process serializes commands and ensures that each individual client's commands are executed in order. It is possible that a command acts on a new object that is in the queue but has yet to be added to the scene graph or that has already a delete command ahead of it in the queue. To prevent these race conditions from blowing up the system, we have used a semaphore locked lookup table of nodes that will be added or deleted the next timer is triggered. Each command checks to make sure that it is not operating on a deleted node or if it cannot find the node in the main scene graph, it checks the add table to see if the node is waiting to appear.

This concept is not perfect, but in practice it has performed very well. There is the possibility of objects being added and deleted before the user could ever see them. However, this ends up being a positive feature, in that when the system is receiving too many commands to keep up, it gracefully drops commands and combines changes. There is no point in trying to change the scene graph faster than it can render.

### 2.1.4 Scene graph conventions

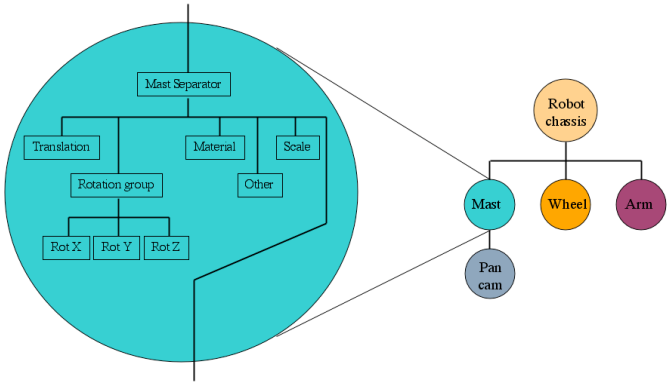


**Figure 2.1.4.1**: Scene graph convention for Viz.

Dealing with a lot of objects can become tedious, especially if their characteristics depend on other objects, such as how the position of a wheel of a robot is usually expressed relatively to robot's body reference frame. Organizing the geometries hierarchically, where each child's characteristics depend on its parents, solves this problem and also optimizes the 3D scene representation for rendering. This tree of object is referred to as 'scene graph' and is implemented in many 3D graphic APIs: VRML, OpenInventor, and WorldToolKit. OpenInventor was chosen as the Viz 3D graphics library because it natively supports the VRML-1 format and offers a variety of desirable features such as draggers for user interaction with the 3D scene and animation objects.

Each Viz object is given a basic scene graph in a standard convention (Figure 2.1.4.1). With this convention, the Viz server is able to find and manipulate objects in a consistent manner. For example, if a client wants to refer to an object in Viz, the object's name must be specified in the message. This means that all objects must have a unique name. Also, special objects such as 'Root' (the parent of the whole scene graph) have predefined names. Below the named object, there must be a translation node, a group of three rotation nodes, a material node, and a scale node. Finally, the object geometry and any attached objects are at the far right of the scene graph.

## 2.2 Split Views, Draggers, and Manipulators

OpenInventor provides several ways of rendering the 3D scene called 'Viewers'. The SoXtExaminer viewer was slightly modified to give the user a view of the scene with the horizon line always horizontal and never upside down. By use of sliders, the user can rotate the point of view to the left or right, up and down or

zoom in and out. Another useful feature is the arrow that designates by mouse-click a feature to be looked at more closely. In addition, draggers and manipulators are OpenInventor objects embedded within the scene graph that allow the user to truly interact with the scene: for example for Mars Polar Lander, we inserted cylindrical draggers into the articulated robotic arm, so that the user could rotate the joints by moving the draggers with the mouse. See Section 3.2 for more details on split views, draggers, and manipulators.

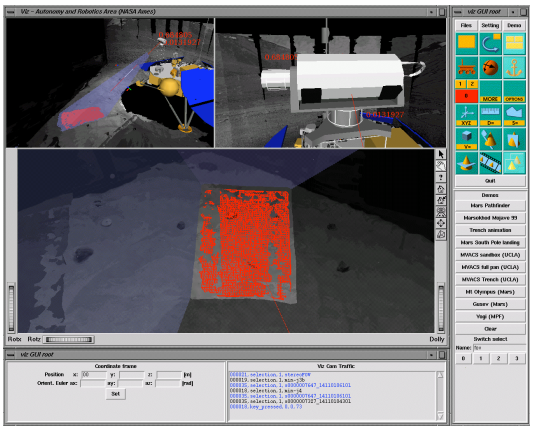

Figure 2.2.1: Planning an imaging sequence for the Mars Polar Lander in the Viz split view mode.

**2.3 X-Core - In Model buttons and exploding views**

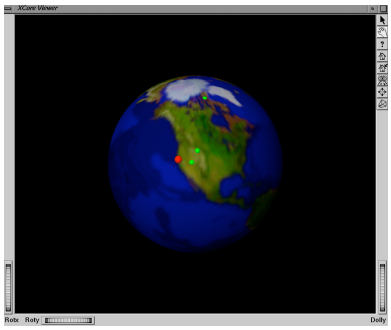

**Figure 2.3.1**: Global database access through OpenInventor model.

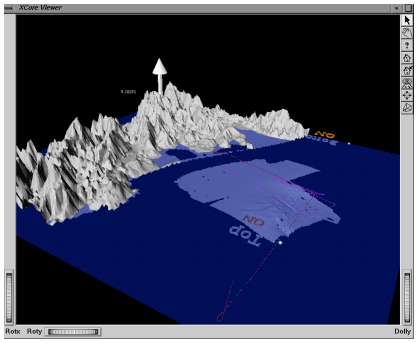

**Figure 2.3.2**: Pop up of a world model from Figure 2.3.2 of the Eel River, northern California. Pictured are a USGS DEM, STRATAFORM EM1000 multibeam, ship tracks, and a movable sea surface level. See Figure 3.4.1 for images of drilling down into more detailed models of cores and seismic lines.

The X-Core extension[1] is designed based on the requirements of marine geologic research. X-Core is composed of 30 small programs that convert most of the common data formats found on research ships into

ASCII Open Inventor/VRML 1.0. X-Core makes two critical changes to the OpenInventor ivview or mainViz programs to provide X-Core functionality. The extension is simple enough that it can be quickly incorporated into any OpenInventor based program. The first modification is to allow touch sensors similar to those found in VRML97 by creating a new URL called "switch". The SoWWWAnchor with this URL allows the user to turn off data layers within a 3D model by clicking on a button within the 3D world. When picked, the SoWWWAnchor node toggles between sub-scene graphs of a SoSwitch node located below in the scene graph. If there are just two sub-scene graphs, one empty and one with a visualization of some data, this will function as an on/off switch. The second change made to ivview was to add a "shell:" URL to SoWWWAnchor. With this change, scene graphs can contain objects that, when clicked, will run an arbitrary shell command.

X-Core takes advantage of the shell URL to create hyper-linked worlds for the scientist to explore. The scientist is first presented with a 3D model of the world (e.g. Earth, Mars, or Venus) with markers for each field site in the database (Figure 2.3.1). When the user clicks on one of the markers, the shell URL is used to invoke another xcore-ivview program with the model representing that field area. For example, clicking on the northern coast of California brings up a model of Humboldt, California and the near-shore bathymetry (Figure 2.3.2). From there the user can click on particular sediment cores collected in the area and drill down to the level of data that user would like to view. This can go all the way down to the microscopic level with data from digital microscopes.

## 3. APPLICATIONS AND RESULTS

### 3.1 DEXterous Mobile Walker

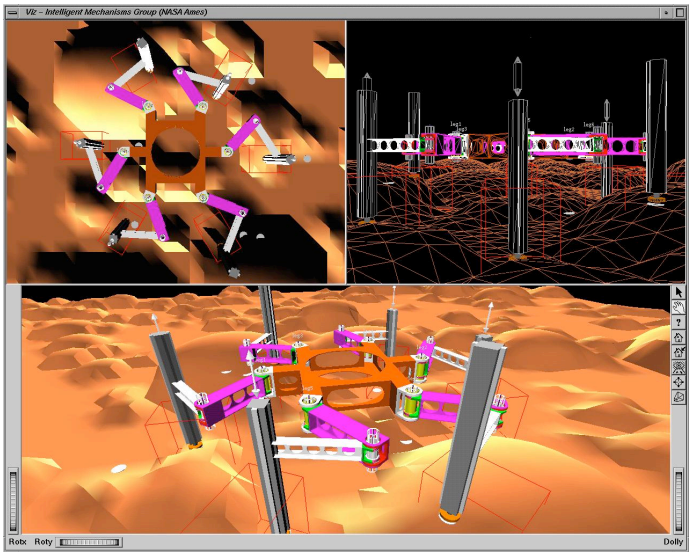


**Figure 3.1**: DEX in his virtual Viz home.

One of the first non-Martian projects that utilized the Viz software was a series of locomotive gate studies performed in 1998 at NASA Ames as prototype for a six leg walker robot named DEX. Prior to building the DEX robot, it was advantageous to visualize ramifications of potential locomotive gates. The Viz software was used to create a complete simulation of the DEX robot, allowing researchers to control the robot as it virtually traversed a variety of terrains (Figure 3.1). Although a physical DEX was not completed, engineers did interface Viz with the DEX robot's servo controllers. Using this interface, commands were sent to the DEX servo control hardware when instructed by user-actuated graphical controls inside the Viz virtual environment.

Working on the DEX project, the advantages of the Viz client server model were quickly evident. It took a pair of engineers only days to create the DEX simulation described above. The software to simulate DEX

was built by first creating an OpenInventor model of the robot. The inverse kinematics of the legs and other control functions were then implemented in the Python programming language. Finally, the 2D portions of the UI were implemented in Python/Tk. The separation of these portions of the software allowed the most appropriate language to be chosen for each task.

### 3.2 Mars Polar Lander

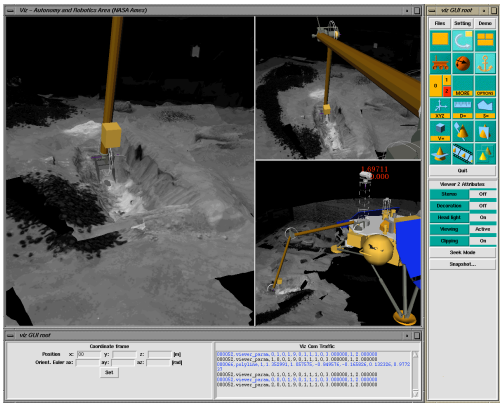

**Figure 3.2.1**: Mars Polar Lander test-bed viewed in Viz while digging a trench during MORT number 5.

The Mars Polar Lander (MPL) was scheduled to land on the Martian South Pole in December 1999. The science payload included a stereo imager and a 4 joint robotic arm equipped with a camera and scoop mounted on the end of the arm. Viz was tightly integrated into the MPL science and operations team at UCLA and included integration with JPL mission databases and Deep Space Network (DSN) configuration. Using Viz, we brought unprocessed VICAR images straight from the database, corrected and calibrated the images, produced 3D stereo meshes with the JPL/Ames (JAMES) pipeline, and automatically integrated the new terrain patches on the fly into the Viz display located in the press conference area (Figure 2.2).

During Mission Operations Readiness Tests (MORTs), Viz proved invaluable with its multiple view mode for planning and analyzing operations where the robotic arm dug a trench in the soil. The science team had to dig at a desired location, place the soil on a discard pile, image the pile and image the excavation with the close-up arm camera (Figure3.2.1). All of this required that the user move the arm into place using OpenInventor manipulators and draggers. The coordinated 3-way split views let the operator see the arm position, have an overall situation awareness, and view the predicted image view that would be acquired from the specified joint angles. Additionally, we embedded calculators attached to 3D text nodes that displayed the joint angles continuously as the arm moved. The operator could then have the UI interrogate the scene graph to capture the joint states at any desired waypoint. The net result is that planning took much less time and accidental imaging of undesired locations was greatly reduced when compared to operational techniques used on Mars Pathfinder.

### 3.3 ROV/AUV live control and data

Although originally designed for environment visualization and robot control during Martian robotic exploration missions, Viz was quickly found useful for terrestrial and submarine projects. The Jeremy Project was the first to apply the Viz software to a submarine environment. The project studied the marine archaeology applications of NASA ROV, stereo imagery and visualization technology, with the goal of making marine archaeology more accessible. The project used a NASA Ames stereovision 3D modeling package to create complex 3D models of underwater objects and terrain by calculating the distance to each pixel in the image plane via a left/right offset sub-pixel-kernel-correlation algorithm (Figures 3.3.1). After the model building software created the 3D models, they were then inserted into the Viz virtual world for visualization and analysis.

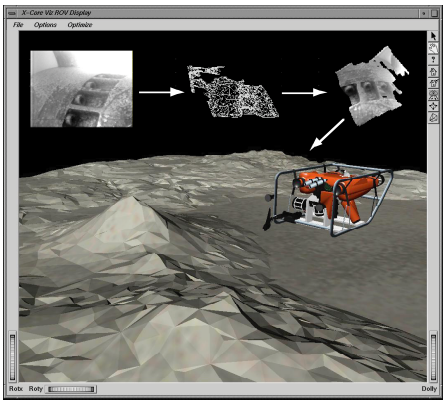

**Figure 3.3.1**: Viz ROV setup presented at the 1997 National oceans Conference in Monterey, CA. The vision processing starts with a stereo pair of images to produce a mesh model. The left image is then texture mapped and imported into the 3D model. The ROV model is fitted with draggers and manipulators to control the pan and tilt of cameras and vehicle position.

During the Jeremy project, Viz was used to visualize and measure underwater artifacts. Although a limited number of models were created, the use of Viz in conjunction with the described stereo modeling software was found to be an effective tool for use in submarine archaeology. The error in linear measurements using Viz and the described stereo modeling package was approximately ten percent. Although this seems a large error, when you consider the difficulty of working underwater and the benefits afforded by these methods, the Viz stereo system is a compelling technology.

### 3.4 TTN136B - The Whole Earth down to microscope scale

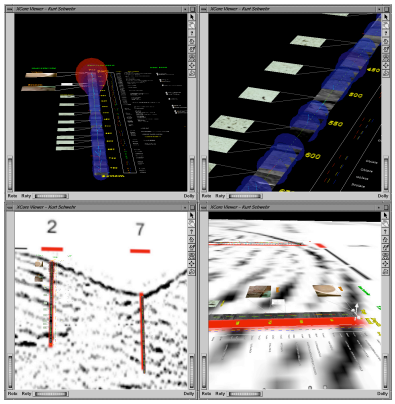


**Figure 3.4.1**: X-Core views of cruise TTN136B. Top two windows are of piston core 02PC, while the bottom two windows show cores 02PC and 07PC registered and placed on top of chirp seismic data.

The cruise TTN136B (Chief Scientist: Schwehr), provided an opportunity to develop the X-Core extension and conversion programs (Figure 3.4.1). This cruise focused around acquiring 7 piston cores on the continental slope in the "Humboldt Side"/Eel River area of northern California. The project required looking at relationships from the kilometer to micrometer scale. X-Core was developed to integrate a mounting pile of data that obscured the relationships between observations and often the tops of many desks. Analysis included microscopic smear slides, paleomagnetic AF demagnetization, anisotropy of magnetic susceptibility, alternating gradient force magnetometry (AGFM), grain size, 14C, 210Pb, & 137Cs dating, core photography, chirp single channel seismics and multibeam sonar. Trying to navigate a deskful of this data becomes a challenge at best.

The traditional method consists of collecting plots on a poster and in a binder of associated material. Ambitious scientists then frequently try to use such programs as Adobe Illustrator to integrate the data into

a spatially coherent layout. Unfortunately, there is so much relevant data that there is never enough available wall space to fit the posters.

With X-Core switches, pop-up sub graphs and URL links, X-Core creates a spatial database referenced to each core's photograph. On the ship, the cores are cut into 1.5 meter sections. The process begins with photographing all of these sections next to each other on a large table. Next the program then extracts and corrects each core segment. Finally, the core is reconstructed as one long picture at the correct scale with no rotational distortion. With X-Core, this process takes just a few minutes from scanning the photo to viewing an OpenInventor scene graph in the local core frame of reference, compared to hours using Adobe Photoshop.

Once all of the core "worlds" are produced, they can be combined with seismic lines and multibeam bathymetry based on the p-code GPS latitude and longitude. Optionally they can be placed next to each other for close examination of inter core relationships.

### 3.5 NBP0209 - Ship integration

The functionality of both Viz and X-Core was first combined for a live research cruise during December 2002 on cruise NBP0209 (Chief Scientists: Cande and Stock) of the RVIB Nathaniel B. Palmer from New Zealand to Antarctica. Integration took just a couple hours. The ship's systems are designed around a Linux based Data Acquisition System (DAS) that publishes all data streams through a keyword coded shared memory access system. A small script was written that grabbed gravity, magnetics, water depth, and GPS coordinates, which are then passed the data on to X-Core/Viz for live viewing. The magnetics data are filtered output from a towed proton-precession magnetometer. A Kongsberg-Simrad EM120 12 kHz swath sonar provided multibeam sonar covering up to 150 degrees of sea floor with 191 beams. Scripts pushed the bathymetry data into the visualization as the data were finished with processing by MB-System and GMT. Such a system has huge potential for reducing the operator load for the scientists on watch, who must currently monitor over ten different displays distributed throughout the science bay. In under a day, all of this data was integrated into a single display in an intuitive view of ship activities. For the test deployment, we used a single SGI O2 with 256MB of RAM.

## 4. CONCLUSION

Due to appropriate choices from existing software libraries and beneficial design decisions, the implementation of a Viz/X-Core based visualization system can be accomplished in a matter of days. Viz/X-Core has proven useful on projects and field missions ranging from visualization and robotic control for planetary exploration and deep-sea research to kinematic simulation for robot design to vast-scale geographic data visualization. For new robot designs, we highly recommend putting the CAD models into a world such as Viz to see how the model behaves prior to “bending metal.”

## 5. FUTURE DIRECTIONS

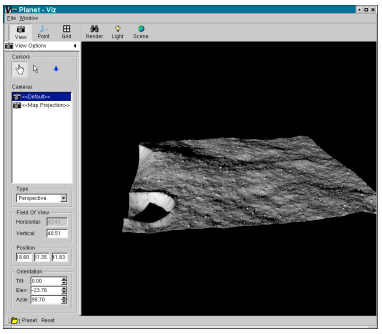

**Figure 5.1**: Shadows in Viz version 2 as viewed on a cratered planetary surface.

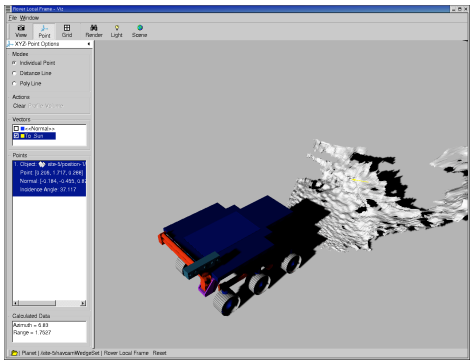

**Figure 5.2**: Mars Exploration Rover CAD model viewed in Viz with stereovision generated terrain patch.

The newest version of Viz at NASA Ames (version 2) supports shadows (Figure 5.1) and is tightly integrated with the Virtual Robot system (Fluckiger, 1998). The Mars Exploration Rover teams will be using Viz as one component of managing and exploring data from the Spirit and Opportunity rovers in 2004 (Figure 5.2). NASA Ames has begun integrating, planning, and scheduling software to produce an overall Mission Simulation Facility. Viz will continue to progress in its distributed operation and collaboration modes that rely heavily on the open network design. See Hesina et al. (1999) for example of one such OpenInventor based implementation.

NASA has already spent considerable work on the human factors of geologists and robotics engineers. For example, McGreevy (1992) outfitted a field geologist with a head mounted display that replaced their normal vision and had two different geologists evaluate a field site. These lessons were brought into VEVI for the TROV interfaces and evaluated during field deployment. We will continue this work through experience with actual oceanographic and Mars based exploration.

With X-Core, we will be looking to more tightly integrate the system with ship operations and data acquisition so that marine data takes a more direct path into accessible databases. Scripps Institution of Oceanography is working on building databases of both multibeam and geologic collections data. As this data comes on line, X-Core will provide an efficient access technology for scientific investigation.

## 6. ACKNOWLEDGMENTS

We would like to thank the many groups that have supported the work on Viz and X-Core: NSF, ONR, and NASA for funding the research cruises; spacecraft missions, and the development of Viz; Cal-IT2 and the SIO Visualization Center for supporting the development of X-Core; L. Tauxe, N. Driscoll, and the entire staff in the Intelligent Mechanisms group without whole this work would never have been possible; JPL and UCLA for allowing us to be a part of the Mars Pathfinder and Mars Polar Lander mission teams; the JPL FIDO and MER teams; Judd Bowman was the primary implementation engineer for Viz 2.x.

(1) The X-Core extension is available for down at http://schwehr.org/xcore/.

## 7. REFERENCES

Fisher S. Virtual Environments: Personal simulations & telepresence. In: Helsel, S, Roth, J (Eds). Virtual Reality: Theory, Practice and Promise. Meckler Publishing: Westport; 1991.

Fluckiger L. A robot interface using virtual reality and automatic kinematics generator, 29th International Symposium on Robotics 1998, DMG Business Media: Redhill; 123-126.

Fonseca L, Mayer L, Paton M. ArcView objects in the Fledermaus interactive 3-D visualization system; an example from the STRATAFORM GIS. In: Wright D (Ed). Undersea with GIS. Environmental Systems Research Institute Press: Redlands; 2000, 1-21.

Hesina G, Schmalstieg D, Fuhrmann A, Purgathofer W. Distributed Open Inventor: A Practical Approach to Distributed 3D Graphics, Proc. of the ACM Symposium on Virtual Reality Software and Technology 1999, ACM: New York; 74-81.

Hine B, Stoker C, Sims M, Rasmussen D, Hontalas P, Fong T, Steele J, Barch D, Andersen D, Milers E, Nygren E. The Application of Telepresence and Virtual Reality to Subsea Exploration, ROV '94, Workshop on: Mobile Robots for Subsea Environments, Monterey, CA 1994.

McCann M. Creating 3-D Oceanographic Data Visualizations for the Web, The Seventh International Conference on Web3D Technology 2002: Tempe.

McGreevy M. The Presence of Field Geologists in Mars-like Terrain. Presence 1992; 1,4: 375-403.

Nguyen L, Bualat M, Edwards L, Flueckiger L, Neveu C, Schwehr K, Wagner M, Zbinden E. Virtual Reality Interfaces for Visualization and Control of Remote Vehicles. Autonomous Robots 2001; 11: 59-68.

Reddy M, Iverson L, Leclerc Y. GeoVRML 1.0: Adding Geographic Support to VRML. GeoInformatics 2000: 3.

Pardo-Castellote G, Schneider S. The Network Data Delivery Service: Real-Time Data Connectivity for Distributed Control Applications, International Conference on Robotics and Automation 1994 (San Diego, USA, 1994), IEEE Computer Society Press: Chicago.

Srinivasan R. RFC 1832, XDR: External Data Representation Standard, August 1995, [WWW Document] http://www.faqs.org/rfcs/rfc1832.html (Accessed July 2003[])

Stoker C, Zbinden E, Blackmon T, Kanefsky B, Hagen J, Henning P, Neveu C, Rasmussen D, Schwehr K, Sims M. Analyzing Pathfinder Data using Virtual Reality and Super-resolved Imaging. J. Geophys. Res., Planets 1999; 104(E4), 8889-8906.

Thomas G. Real-time panospheric image dewarping and presentation for remote mobile robot control. Advance Robotics 2003; 17,4: 359-368.

Wernecke J. The Inventor Mentor. Addison-Wesley; 1994, 514 pp.

Ware C, Plumlee M, Arsenault R, Mayer L, Smith S, and House D. GeoZui3D: Data Fusion for Interpreting Oceanographic Data, Oceans 2001 (Hawaii, USA 2001).