AFRL-HE-AZ-TR-2000-0163

UNITED STATES AIR FORCE RESEARCH LABORATORY

INTELLIGENT TRAINING DEVELOPMENT METHODOLOGIES

Randy J. Stiles

Lockheed Martin Missiles & Space Company
Research and Development Division
3251 Hanover Street
Palo Alto, CA 94304-1911

July 2001

AIR FORCE RESEARCH LABORATORY
HUMAN EFFECTIVENESS DIRECTORATE
WARFIGHTER TRAINING RESEARCH DIVISION
6030 South Kent Street
Mesa AZ 85212-6061

NOTICES

Publication of this report does not constitute approval or disapproval of the ideas or findings. It is published in the interest of STINFO exchange.

Using Government drawings, specifications, or other data included in this document for any other purpose other than Government-related procurement does not in any way obligate the US Government. The fact that the Government formulated or supplied the drawings, specifications, or other data, does not license the holder or any other person or corporation, or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

The Office of Public Affairs has reviewed this report, and is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This report has been reviewed and is approved for publication.

JAMES L. FLEMING
Project Scientist

DEE H. ANDREWS
Technical Director

JERALD L. STRAW, Colonel, USAF
Chief, Warfighter Training Research Division

Direct requests for copies of this report to:

Defense Technical Information Center
8725 John J. Kingman Road, Suite 0944
Ft. Belvoir, Virginia 22060-6218
http://stinet.dtic.mil

Copies Furnished to DTIC
Reproduced From
Bound Original

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
July 2001	Final	Apr 1993-Jan 1996

4. TITLE
Intelligent Training Development Methodologies

5a. CONTRACT NUMBER
F41624-93-C-5000

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62202F

6. AUTHOR(S)
Stiles, R.J

5d. PROJECT NUMBER
1123

5e. TASK NUMBER
A3

5f. WORK UNIT NUMBER
11

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Lockheed Martin Missiles & Space Co.
Research and Development Division
3251 Hanover Street
Palo Alto, CA 94304-1911

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory
Human Effectiveness Directorate
Warfighter Training Research Division
2509 Kennedy Circle
Brooks AFB, TX 78235-5118

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-HE-AZ-TR-2000-0163

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
Air Force Research Laboratory Technical Monitor: Dr. James Fleming, Brooks AFB, TX

14. ABSTRACT

This research effort developed an immersed virtual environment for the instruction of orbital mechanics and Training Studio, a more general virtual environment instructional authoring system. Training Studio provides the instructor with the capability of developing instructional simulations and not needing a computer programmer to translate instructional objectives into code. It provides the instructor with a visual programming system that operates the virtual environment. In a Training Studio session, there can be several processes handling interaction and display of the virtual environment (Vista Viewer), several processes modeling the understanding of the participants (Training Assistant), and several processes controlling the simulation and visual aspect of individual objects (Agent Framework and Profiler). Suggestions for further evolution of this type of software are provided.

15. SUBJECT TERMS
Instructional Simulation; Intelligent Training; Intelligent Tutoring System; Virtual Environment; Virtual Reality

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. James Fleming
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL	45	19b. TELEPHONE NUMBER *(include area code)* (210) 536-2034 DSN 240-2034

TABLE OF CONTENTS

Summary

This is a final report of the research and development performed by the Lockheed Martin Advanced Technology Center on the Intelligent Training Development Methodologies (ITDM) contract F41624-93-C-5000. The funding agency for ITDM was the U.S. Air Force Armstrong Labs Technical Training Research Division, and the period of performance was from April 18, 1993 to January 15, 1996. During this period, the Lockheed Martin team delivered on the goal of an immersed virtual environment for the instruction of orbital mechanics, developed software releases for a more general Virtual Environment instructional authoring system (Training Studio), provided current and topical research information to the funding agency, and earned a following effort to apply this technology to Defense Department needs as part of a Focused Research Initiative.

The Training Studio is a networked, immersive virtual environment developed for the purpose of training. To support development and delivery of instructional content in a virtual environment, we identified several goals for the Training Studio. Primary among these was to adopt an overall metaphor allowing the instructional developer to build instructional content as directly as possible. This metaphor is the Training Studio: a stage, audience and control room in a virtual environment that the instructor uses to develop courses, and the place where instruction is delivered. Supporting goals follow:

- Multiple participants, including instructor and students interacting
- Definition of scenes and simulations using the virtual environment
- Immersive and through-the-window virtual environment interaction
- Individual instruction using an intelligent tutoring system
- Support a blend of guided and exploratory instruction

There are a number of technologies developed or advanced under the ITDM contract which we feel contribute to an effective virtual environment for training. Foremost among these was the architecture approach using the virtual environment as a display server, and developing a network protocol that supports updating many such VE display servers. The philosophy behind this approach was to abstract the creation and use of distributed, multi-participant 3D spatial scenes in such a way as simplify instructional development, and yet still provide required flexibility.

With the Training Studio software, we have realized a software system in which it is possible to construct and deliver training scenarios in an immersed, interactive setting. The proposed final goal of an orbital mechanics demonstration was achieved early in the course of contract and we went on to build a more general system. This more general system will see its fullest application in the Virtual Environments for Training program, where the generality of separating viewing & interaction from simulation will be used to integrate the RIDES and SOAR systems. We see the Training Studio software in its final delivered form as an Internet-capable suite of software for developing and delivering immersive simulation-based training in a virtual environment. We see training content being developed at sites like the TTRD, essentially VE Training Centers, by domain experts and technical assistants, and then being fielded using the world-wide-web protocols to other Defense Department sites in a time-efficient and responsive manner. Changes to curriculum, simulations and 3D models can be distributed quickly. We feel the Training Studio going beyond what is available in current world-wide-web tools such as flat screen VRML browsers by supporting true immersive, real-time interactions, and by supporting the kinds of instructional abstractions we have developed under ITDM.

PREFACE

This research was performed for the Air Force Research Laboratory, Human Effectiveness Directorate, Warfighter Training Research Division (AFRL/HEA) (formerly Armstrong Laboratory's Technical Training Research Division) under USAF Contract No. F41624-93-C-5000 with Lockheed Martin Advanced Technology Center, and Work Unit 1123-A3-11, Advances in Virtual Reality. The Laboratory Contract Monitor was Dr James L. Fleming, AFRL/HEAI, at Brooks Air Force Base, TX.

Documentation of this research was delayed due to personnel reassignments and the laboratory reorganization. The original technical monitor was Dr Fleming; however, the final administrative work necessary to publish this report was accomplished by Dr Donald L. Harville, AFRL/HEAI, at Brooks AFB TX.

Introduction

This is a final report of the research and development performed by the Lockheed Martin Advanced Technology Center on the Intelligent Training Development Methodologies (ITDM) contract F41624-93-C-5000. The funding agency for ITDM was the U.S. Air Force Armstrong Labs Technical Training Research Division, and the period of performance was from April 18, 1993 to January 15, 1996. During this period, the Lockheed Martin team delivered on the goal of an immersed virtual environment for the instruction of orbital mechanics, developed software releases for a more general Virtual Environment instructional authoring system (Training Studio), provided current and topical research information to the funding agency, and earned a following effort to apply this technology to Defense Department needs as part of a Focused Research Initiative.

Other research groups have constructed significant demonstrations which realize virtual training environments in areas as diverse as sea navigation [Magee95], ship-board damage control [King95], andspace repair missions [Loftin95]. These demonstration systems provide empirical evidence that virtual environments can be successfully used as part of structured training, and some controlled studies indeed indicate that virtual environments can be effectively used for training and mission rehearsal in spatially complicated areas such as shipboard firefighting [Tate95]. The ITDM effort, however, was one of the very first research and development efforts to address issues for authoring and delivering training in a virtual environment, developing methodologies and software that support delivering instruction in a virtual environment [Grant91]. This report summarizes the underlying approach, technical insights, and critical technologies developed during the course of the ITDM contract.

Methods, Assumptions & Procedures

This section discusses the task structure for the ITDM effort and what was accomplished under each task, introduces the Training Studio operating concept under which development progressed, and relates the resulting system specification.

Contract Task Structure

The Intelligent Training Development Methodologies research effort was conducted over the period of roughly two and a half years. To report to the reader what was accomplished during that period, it is essential to refer to the original proposed task structure as an outline of the effort.

The research and development activities that comprised the ITDM efforts were categorized into three separate tasks: Requirements Analysis, Systems Design, and Demonstration. Both Task 1 (Requirements Analysis) and Task 2 (Systems Design) included a summary report of findings; Task 3 (Demonstration) provided a working demonstration of the application software for the sponsoring agency's review and evaluation. Task 1 was completed early in the contract since findings from the Requirements Analysis drove the design of both the system and demonstration tutorial. Initially, Task 3 was scheduled to begin after the bulk of the systems design was complete. Instead, the development of the demonstration commenced shortly after Task 2 began. The rationale for this change was to allow early use of the prototype system supporting identification of user needs. A summary description of each of the three tasks follow.

Task I: Requirements Analysis

A requirements analysis was conducted first to ensure that the design of the system met the requirements of an instructional authoring and delivery system for virtual environments and incorporated the capabilities of existing hardware and software. To accomplish this task, the following activities were performed:

- Reviewed the literature in advanced training methodology to assess the likely impact of key proposal concepts:
  - TV studio paradigm as a VE extension to a conventional learning setting
  - Scripts as a metaphor for VE-based lessons
  - Visual programming for courseware design and development
  - Courseware delivery with eye objects, text windows, and voice balloons
- Scanned the VE commercial and research literature for relevant technology in:
  - Networked virtual environments
  - Scripting and managing virtual environments
  - Agent-level network communication protocols
- Reviewed existing courseware systems with an eye to integration with VE systems
- Interviewed Lockheed Technical Operations Company (LTOC) training personnel to determine key needs facing curriculum designers, developers and instructors in the example orbital mechanics domain.
- Arrived at a set of requirements for the final software system which covered six areas: agent class systems, physical dynamics, visual languages, visual work spaces, distributed communications, and graphical environments [Stiles93a].

The findings from the above were reported in a System Requirements Analysis document. Both draft and final versions of this document was delivered early to the funding agency to ensure that the funding agency;'s concerns were addressed in the system requirements.

Task II: System Design

The design of the instructional system was based on the results of the System Requirements Study, and incorporated virtual environment tools (VE), concepts and architecture that address the stated needs of designers, developers, and instructors. Modifications to the design were made based on insight gained from working on the orbital mechanics domain, customer input from technical and system reviews, and experience during development. The following activities were accomplished under this task.

- Analyzed key system requirements that drive design (e.g., response time)
- Specified and designed instructional VE toolkit capabilities
  - Studio Control Room, Stage and Audience
- Specified, designed and developed courseware toolkit capabilities
  - Script/lesson template; student model; networked protocol
- Specified, designed and developed VE support capabilities
  - Script execution; VE interactions; 3D device interfaces; 3D rapid rendering system
- Formulated a comprehensive, distributed architecture capturing the above
- Evaluated the system design against the initial requirements
- Developed and delivered initial alpha-level software for evaluation by technical representatives of the funding agency in October of 1994.

The findings from the above were reported in a Concept Design Document, also referred to as the Product Specification Report. This report covered what capabilities a final product should contain.

Task III: Demonstration

The team defined, built and executed a series of demonstrations of the System Design Concept as a deliverable under the contract. The demonstrations integrated existing VE interaction hardware and software with existing satellite visualization software to teach a set of selected lessons in orbital physics. Each demonstration showed increased functionality, based on input from the customer during the previous demonstration. The demonstrations took place at Lockheed Artificial Intelligence Center in Palo Alto, CA. To support the demonstration task, the following activities were accomplished.

- Built the instructional scenario to be demonstrated
  - Defined a scenario for teaching elements of orbital mechanics
  - Refined the scenario into a sequence of lessons using the studio concept
  - Invented VE interactions for the instructor to create the lessons as scripts
  - Created agents representing the student and the orbital objects
- Integrated existing Toolkits
  - 3-D stereo visual and audio device interaction
  - Orbital Mechanics simulation
- Assembled the demonstration
  - Built the control room and equipped it with the necessary scripting tools
  - Built the stage and audience views
  - Built and tested the instructor's and student's Virtual Environments
- Delivered further alpha software releases
  - Responded to system evaluations and suggested changes by incorporating them into Training Studio software.
  - Delivered User's Guide document explaining use of system
  - Responded to bug notices by delivering bug fixes in a prompt manner.
- Supported an additional demonstration application
  - Revised system functionality in support of ground controller training system needs

Training Studio Operating Concept

There are two primary modes of Training Studio usage. The first of these is for development of instructional courses taking place in a virtual environment. The second is the delivery of these instructional courses to one or more students.

For instructional development, an instructor only has to use a subset of the Training Studio components. A development session consists of one virtual environment viewer (Vista), one intelligent tutoring system in editing mode (Training Assistant), and one agent simulation framework (Smalltalk). The instructor uses Vista to define scenes, the Training Assistant to structure the domain model, course model, and student model, and the simulation framework to define and run simulation agents that are part of the instructional content.

For instructional delivery, there are a number of choices available. A training session can be conducted with or without an instructor, and with one or more students. One student can use the Training Studio to learn a topic. An instructor can view a training session where one or more students are learning, for the purposes of intervention, or to monitor the use of a newly developed course. Or several students can be part of a training session where no instructor is present. In these sessions, each student is monitored by a Training Assistant which will take the place of an instructor.

In all of these cases, the Training Studio can be thought of as the place where the participants gather. Each Vista Viewer is a window onto the action taking place in the Training Studio. Assistant agents, such as the Supervisor, Profiler, and Training Assistant, help the

participants use the Training Studio, and simulation agents provide instructional content in the Training Studio.

Authoring as Important as Delivery

The Training Studio was designed under the premise that providing the instructional developer with the capability of developing instructional simulations could more efficiently produce (more effective) training, rather than relying on a computer programmer to translate instructional objectives into code. An authoring capability, then, was required to allow a non-programmer to develop the instructional sequences. To allow the developer to experience the lesson from the student perspective during development, the Training Studio provides a visual programming system that operates within the virtual environment. Providing this authoring capability to the developer facilitates the modifications, fine-tuning, and experimentation of instructional sequences that may not be made if the changes must first be articulated to a programmer, who must translate the perceived modifications into code, the results of which are evaluated by the developer. Additionally, the ability of the instructor to easily modify scenarios facilitates the reuse of existing courseware from current or previously developed courses. The instructor can simply load the lesson, make necessary changes by manipulating the components, then save the new version for current use.

Networked Immersion

One of the goals of the ITDM effort was to support multiple participants during both team training and group instructional sessions. Key to realization of this goal was the networking of students and instructors to allow all participants to experience instruction within a shared environment. This allows all participants to simultaneously observe the same activity, thereby providing a method for one instructor to work with a group of students.

Networked immersion also allows all participants to view the actions of other participants as well as experience the implications of those actions. Again in a group instruction situation, this capability can be used for class observations of another student carrying out a task, and observing the consequences of the actions. On a team training level, this shared experience is essential for realistic training situations. Each team member can not only observe the activities of others, but can also view and, if necessary, respond to the results of those actions.

Organizing Metaphor

Our approach for both development and delivery of instruction centers around a training studio metaphor. An instructional developer initially sees a training stage, audience section, and studio control room. The stage is where action takes place, the audience where observation takes place, and the control room is where definition takes place. Any simulations relevant to instruction take place on the stage. The tools used by the instructional developer are present in the control room, and during development, the audience section is used as a representation for the kinds of students that will be present at training sessions. During delivery, the audience section includes any students participating in the system, the stage is their central focus, and control room is not visible to them.

Stage

The Training Studio stage is a visual point of reference where instructional activity takes place. Rather than confronting the instructional developer with a blank void, the stage is a plane where simulation action and student interaction occur. For everyone using the Training

Studio, whether in instructional development or delivery mode, the stage serves as an anchor - a point of reference.

As part of the instructional development process, the developer defines scenes by placing simulation objects at given starting positions on the stage. For instance, to show a location such as an airport, the instructional developer can place the buildings and landing strips on the stage to arrive at a good representation. The developer can also place student representations in initial positions, to indicate they are to be active participants in the exploratory simulation, and can change the environment associated with the stage to show fog, different times of the day, stars, stage colors, or explanatory stage textures.

For delivery of instruction, the immersed student sees the stage as a point of reference for orienting themselves, and as a visual reminder of where to look during instruction. Student placement on the stage means exploratory interaction is expected from the student for that instructional scene. With the proper texture applied to the stage floor, they can look down and quickly judge where they are in the virtual environment.

Audience

The Training Studio metaphor provides for student observation via an audience section, represented by a set of seats that can be constructed near the stage.

During development of a given scene, the instructor can select a student representation, and then a seat, to indicate that during that scene, that student should be an observer. During instructional delivery, the student is constrained to that position, but can look around. When in the audience, the students cannot select any studio objects with any effect. This is especially useful when the instructor needs some students to learn from the actions of another set of students, or if in general the simulation should not be altered during the course of an explanation. The audience section is also a good way to deal with students as a group when defining the action that must take place in a scene. The audience as a whole can be moved along a path, or transported to a given area in the studio.

Control Room

The tools required for building an instructional simulation or sequence of activities are found in the Training Studio control room. This is a floating platform with control panels that can be moved around the stage by the instructional developer. Different types of control panels exist for controlling simulations, changing training scenes, and manipulating graphical objects.

Most types of object manipulation can be carried out using controls found in the control room. Prototypes of objects or geometry files can be selected to instantiate a new instance of a graphic object, and changes can then be made to its color, placement, orientation, scale, and place in the scene graph.

A control panel exists for controlling aspects of simulations. Individual simulations taking place on the stage can be toggled on and off, and the rate at which they run can be changed. The abstraction we use to represent this kind of control is that of a simulation clock.

There can be several people involved in training scenarios. We call these participants, and there are control panels for controlling attributes related to participants. The role of a given participant can be defined as instructor or student. Paths through the training studio which one or more participants can travel along can be defined, including their orientation, and the tracking of moving simulation objects. From the control room, student participants can be placed in the audience section, which changes their role to be observing students.

A control panel exists for scene control, which primarily wraps up scene capabilities in the underlying SGI Performer software. Environment controls such as fog and time of day are available here. Toggles which turn on and off the stage cover, stage, stars, and other training studio elements are part of this control panel. The miniature stage control can be considered part of this set of controls.

Results and Discussion

This section discusses the critical technologies developed under contract, the lessons learned during the conduct of the contract and plans for future development, as well as those items delivered during the contract.

System Specification

There are two primary modes of Training Studio usage. The first of these is for development of instructional courses taking place in a virtual environment. The second is the delivery of these instructional courses to one or more students. For a description of these modes please refer the Operating Concept section of this document.

Various software components are available to realize development and delivery in the Training Studio framework. In a Training Studio training session, there can be several software processes handling interaction and display of the virtual environment (Vista Viewer), several processes modeling the understanding of the participants (Training Assistant), and several processes controlling the simulation and visual aspect of individual objects (Agent Framework & Profiler)

Agent Framework

The agent framework used in the Training Studio is based on Smalltalk classes of agents. After initial experiments using a C++ framework, we arrived at the conclusion that a class-based, interpreted framework was crucial to support the on-line, immersive development of training scenarios. Our choice for this framework was GNU Smalltalk. In cases where software performance is paramount, we still use C++, as in the case of visual display and numerical routines. But for rapidly modifying the system we largely rely on Smalltalk.

Class Structure

The base class for most of the Training Studio objects is the Graphic Object class. It is an abstract class, in that there are no instances of Graphic Objects used as part of the Training Studio, only subclasses. In this way we bundle the functionality that is to be common across all objects.

The main categories of subclasses that are derived from Graphic Objects are: object forms, interface objects, programming forms, and agent objects.

Object forms are the actual graphical representations. A box, cone, or cylinder are an examples of graphical representations. Interface objects are objects which structure interactions with Training Studio participants. They may be made up of one or more graphical representations. Examples of interface objects are sliders, buttons, grab buttons, etc. Programming forms are those which represent elements of the programming environment itself. Examples of this are those taken from Lingua Graphica - class, method, and instance representations. Agent objects are those which can act on their own in response to the state of the Training Studio virtual

environment. We make a further distinction between abstract agents and simulation agents, in that abstract agents do not have a physical representation in the physical world, while simulation agents do.

Graphic Objects

We collectively refer to most of the object instances in the agent framework as graphic objects, because they all have inherited the methods and state variables available to the Graphic Object class. The state includes 3D position and orientation, scale, color, form, switch flag, direct callback, and owner, etc. There are appropriate methods for setting or retrieving the value of these states.

Methods also exist as part of the Graphic Object class for transmitting object values onto the communication bus, to networked Vista Viewers. An important concept is that simply initializing a new graphic object does not mean that participants in the Training Studio can see it. For this to happen, an appropriate method for initialization must be called, and this in turn transmits state variable values onto the communications bus, where each networked Vista Viewer can receive them and display their results. The simplest Smalltalk method for initializing instances of graphical representations is *init0: <owner>* where the owner is the object which created this particular graphical object. To initialize more complex graphic objects, we typically use the method *buildForm* which takes no arguments. Most often, buildForm creates one or more instances of graphical representation, sets their owner, placement, and callbacks up, and makes sure they are initialized across the communications bus.

Callback Mechanisms

The abstraction of callbacks is very useful in interface systems, where action is taken in response to a user's choices. The idea is that some graphical form in the virtual environment represents the evaluation of Smalltalk source code that delivers a kind of needed functionality. When the user selects that object, the object calls back the method or function that is associated with the object. Windowing interfaces such as Motif and XView use this idea extensively.

Every graphic object in the agent framework has a slot which can hold a block of Smalltalk code. When an object is selected (picked) in the virtual environment, the owner object slot is not empty and this direct callback slot has a block in it, the graphic object invokes the direct callback with the proper arguments. These arguments are the object that was picked and the owner object. Using these arguments, the code in the block can find out more about the picked object, such as where it was picked in local coordinates, the location of the picked object, what color it is, which participant caused the pick event, etc. Additionally, the callback block can be formed using other object variables in place in the scope of the creating method.

This callback mechanism is used in the agent framework to build buttons, set buttons that collect a set of picked object, sliders that manipulate numerical values, and grab buttons that grab all other pick events for objects and use them as arguments to its own callback functionality.

Graphical Representations

Graphical representations are those objects which can be seen in the Training Studio. They are sometimes referred to as graphical primitives. For each graphical primitive supported in the Vista Viewer software, there is a graphical representation in the agent framework. By instantiating a new graphical representation, such as GOBox using the new method, and then

sending it the message *init0: <owner>*, a graphical primitive is built in all the Vista Viewers connected to the same communication bus as the agent framework.

All graphical representations can be deleted from the Vista Viewers using the *delete* message, and can be rebuilt using the *rebuild* message. Both deleting and rebuilding do not change the representation in the agent framework, rather, they change all the graphical primitives in the networked Vista Viewers to change.

There are graphical object classes in the agent framework for boxes, cylinders, cones, irregular cylinders (conic frustums), angle wedges, lines, circles, rectangles, polygons, gridded surfaces, points, and triangles. Essentially this representation mirrors the functionality present in the Vista Viewer in raw form, and goes on to make instantiating and modifying them fairly easy. For all these graphical representations, through object methods, one can change their color, scale, coordinates, switch setting, parent in the scene graph, and transparency. Most types of graphical representations have additional messages that allow you to change other attributes, such as radii and number of sides.

Interface Objects

A number of interface objects are available as part of the agent framework. These interface objects are collections of graphical representations which are visible in the Training Studio, and an associated group of methods that carries out the interface actions. They are usually a part of the control workspace in the Training Studio, directly inheriting its coordinates in the Studio. An alternative place for interface objects in the scene graph is as a part of the user's view. Support has been built into the Vista system for attaching arbitrary objects to each participant, by way of a participant transformation. This transformation is updated for each user independently, and is guaranteed to always be in front of them no matter where they move their view. One should be careful to avoid obstructing the participant's view with these interface objects, but having this set of interface objects consistently at hand and visible can be useful when immersed.

Vista Viewer

Vista is a networked virtual environment display allowing immersion and networked operation. One or more Vista Viewers give participants a way of interacting with the whole system which we call the Training Studio. The Vista Viewer has been built from the start so that there is a Vista viewer process for each participant in a training exercise. A participant's Vista viewer acts much like an X windows display server - it displays the results of TScript messages arriving across the communications bus, and it broadcasts the actions of the participant using the Vista viewer out onto the communications bus. Though some elements of physical dynamics for objects in the virtual environment are handled in Vista, our emphasis has been to simulate agents and objects over the network and to support a general form of display and immersion in the Vista viewer. By moving any domain-specific content out of Vista and focusing on functionality common to a range of instructional needs, we have created a component useful to instructors in varying domains without a huge degree of re-design.

Though some elements of physical dynamics for objects in the virtual environment are handled in Vista, our emphasis has been to simulate agents and objects over the network and to support a general form of display and immersion in the Vista viewer. This lets us extend the system fairly easily, and the virtual environment system itself is not burdened with the extra computation that simulation can require. Vista is integrated with Distributed Interactive Simulation support and can act as a stealth observer, or broadcast the location of its participant to make him an active part of a DIS exercise [Stiles93b].

Geometry Capabilities

The graphics requirements for Vista are three-fold; first there must be a high degree of realism to support recognition of the same items in the field; second, there must be support for appropriate abstraction and simplification of graphics to allow the student to focus; and finally, there must be the capability to dynamically create geometry of the exact size and form to visually explain dynamic phenomena. Added to this is the need to support these capabilities in a dynamic real-time environment.

To aid in the explanation and instruction of concepts in a three-dimensional setting, the Training Studio system has built in support for graphic objects of different sizes, colors, and shapes, as well as geometric models built with outside modelers and loaded in as files. Vista supports multiple graphic formats to realize the above goals and to reduce development cost. Standard CAD and solid model files can be imported into Vista as a starting place for graphic development.

The dynamic graphic primitives supported in Vista are: boxes, cones, cylinders, irregular cylinders, superquadrics (ellipse-based forms which include spheres), lines, disks, triangles, planes, arcs, irregular polygons, polygon meshes, 3D text, wireframe circles, spheres, wireframe spheres, and points. The various parameters for these objects, such as color and size, can be changed during run-time, and where appropriate, the number of sides can be changed, e.g. a cylinder can change from 4 to 100 sides.

Interaction

In the Training Studio, there are three main manipulation tools supported for input from the user. These are the three-button mouse, the immersive ray cursor, and an immersive glove, based on the Virtual Technologies Cyberglove. All of these manipulation tools support object selection, and these selection events are broadcast using TScript to other processes where the corresponding action results.

The ray cursor is a pointing device that acts much like a laser pointer. One of the advantages over the typical glove approach is that it allows objects to be selected at a distance without having to move yourself close to them. The first object to intersect the ray in the user's view is the one for which a selection event (vrPick) is sent out. If the user keeps the left button down on the ray cursor they will be able to move the object around as though it were stuck on the end of the ray. The ray cursor can be used to fly as well. By pressing the middle button you can change between fly and select modes. The name of the current mode is displayed in the upper left corner of the left eye. Fly mode when immersed is based on where you are looking. If you press the left button you will fly forward in the direction you are looking. If you turn your head while flying, you will fly in that direction. If you press the right button, you will fly backward.

The Cyberglove is supported for selecting objects in the three dimensional space. The Cyberglove is represented visually in the virtual environment as a stylized human hand. This hand geometry directly maps to the Cyberglove worn on the users hand Besides selection using fingers, the ray cursor selection functionality is supported as a ray from the top of the wrist.

Audio is another interaction modality supported by the Training Studio. This conventional audio can be used in two forms in the current system. It can be used as a means of presenting spoken information or for audio sound effect cues related to the occurrence of events in the environment. Verbal presentation can involve an instructor's recorded explanations or audio cues can be used to indicate collisions or bring the students attention to important changes in

parts of the instructional simulation. There are two flavors of verbal presentation supported in the Training Studio: recorded speech and speech generated from text files.

Participant Modeling

The Training Studio system can accommodate one or more networked participants. Each participant, or person, using the system, has a representation inside the Training Studio virtual environment. To keep these representations accurate, so that one participant can perceive what the other participants are doing, we model the position, orientation, and the kinds of roles that each participant has in training scenarios. During training involving more than one participant, icons representing the other people move and track with them to give participants a good idea of where the others are, and what they are doing.

One of the things we found problematic in virtual environments is that people get lost easily in complex scenes. This is often due to the narrow field of view they experience when using immersive devices. Narrow fields of view are a current hardware limitation, and often a performance compromise. Also, even if they know where they are, they can miss crucial events in a dynamic environment and must sometimes be guided. Our work has focused on providing a good blend of free exploration and guided assistance while immersed in the virtual environment. Vista currently can carry students on paths through areas for familiarization, and can automatically point their views at dynamically moving objects, as a way of focusing their attention to specific events.

Vista supports this kind of instructor-directed control with the use of the viewpoint path primitives. These primitives allow several different attention focusing (viewpoint) transitions, and are similar in some ways to capabilities required for Stealth observation in DIS settings [6]. These transitions are:

- **path-orient** travel from point-A to point-B looking straight ahead down the path. This type of transition is useful when explicitly guided paths and viewing are required and is most commonly what we consider an animated path.

- **angle-orient** transition from point-A look ingout with a specified viewing orientation then transition over the course of travel to another preset viewing orientation at point-B. This transition is similar to the previous in that the student's viewpoint is controlled at the beginning and end of the path. It provides more flexibility to the student during movement since, although the path itself is fixed, the student can view anything along the path.

- **object-orient** transition from an initial view orientation set to look at an object in the environment to looking at another (or the same) object at the end of the path of travel. For example, if a maintenance task requires reading a gauge that is in an obscure location, the path might start from a distant position viewing the entire object, then move in closer, constraining the viewpoint on the target location until the gauge comes into full view. The student remains passive. The difference between this transition and the previous is that the instructor specifies student views of an object or objects, (as opposed to locations.) This is useful when the instructor cannot be certain of the exact location of an object (for example, if it is a mobile object) travels along the path.

- **angle-object-orient** transition. A variation which allows the viewpoint orientation to transition between a preset orientation and an orientation looking at an object, where either of these can be the first or the last orientation in the path.

We have built support for other useful forms of immersed participant viewing control. Immersed viewing is complicated by the necessity to allow the participant free motion for

orienting their view. It can be disconcerting to someone immersed when all aspects of head motion has no effect on what is seen. The mismatch between what they is seen and what is expected is too great.

In one form of participant viewing control, called a relative move, a TScript message can cause a named participant to be teleported to with a radius of a named graphic object, such that they will be near the object and viewing the object. The TScript message *vrPartMoveRel* causes this teleport. In another form, the named participant can be tethered to a moving object so that always have the object in the center of their view. The TScript message for this is *vrPartSphereMove* and as the immersed participant changes their head orientation, it causes them to move in a sphere around the tethered object, so that the object is always in front of them. This is an excellent way to let an immersed participant examine a moving object, such as a DIS entity.

Training Assistant

The Training Assistant is an intelligent tutoring system which monitors an individual student using the Training Studio. An instructor builds a concept graph which represents a given domain, and an experts understanding of it. Then the instructor develops a course graph consisting of lessons and lesson units which serve to instruct the student. Each lesson unit can consist of explanations and evaluation tasks which provide updates to its model of the student's understanding. One or more scenes contained within a lesson unit are used to define the simulations and visual state of the networked virtual environment, as a precursor to carrying out the lesson unit.

As a student witnesses explanations, or carries out evaluative tasks, in the virtual environment, the concept graph is updated to reflect his understanding. In cases where understanding is not sufficient, further explanation and evaluation occurs as a form of remediation.

Each scene definition consists of script elements which determine the state of the viewers, objects, and simulations in the Training Studio. These script elements are collectively known as TScript [McCarthy93]. Using TScript, new graphical objects can be instantiated and transformed in the visual scene, as can the participants.

Profiler

Vista includes mechanisms for the dynamically managed complexity of the graphic elements of the environment being experienced. Some of this capability stems from our use of SGI Performer graphics libraries. Performer supports four means of reducing geometric complexity: back-face culling of polygons, distance-based level-of-detail culling, culling based on the viewing area (culling by viewing frustum) and support for partitioning of the geometric database and switching between these partitions [Rolf94].

Added to this is the use of the Profiler agent in combination with other domain specific agents. By using knowledge about the relative importance of objects in the environment as they relate to a given participant, the Profiler applies one of several visual techniques to either reduce the complexity of the rendered object or suppress the rendering of certain objects altogether.

The Profiler is used by the instructional developer both to reduce scene detail for instructional focus and as an effective means for the instructor to optimize rendering performance. The Profiler has several scene capabilities. The first of these is structured scene editing while using the Vista viewer. An instructor can select objects in the environment that should be dropped out

during a given instruction scene, as well as change the level of detail for one object or a class of objects. An easy to use mechanism for selecting previous scenes as a starting point is provided so that this task is not tedious. Currently the specification of scenes is determined off-line, and the Profiler adjusts them in Vista according to these specifications.

The profiler can be used to change the transformation attributes of objects in the Studio, as well as locking them in place. These options can be found under the edit menu. Each option uses the most recently selected object in Vista as its argument, or an object selected from the Profiler catalog.

LG Parser

The Lingua Graphica parser is the agent responsible for handling modification of agent behaviors in the Training Studio. Lingua Graphica is a visual programming language for virtual environments [Stiles92]. Initial prototype versions of this language have been developed under the ITDM contract, but because customer emphasis was placed on responding to system evaluation, and we worked with finite resources, we put Lingua Graphic development on hold. It will be resumed under the follow-on VET effort.

Three-dimensional, solid language constructs are used to visualize and compose programs and higher-level scripts while using a virtual environment. This is a departure from normal two-dimensional visual programming languages, and a departure from the normal way that virtual environments are developed and used. Using Lingua Graphica to specify the behaviors directly associated with virtual objects should decrease the time required to create virtual environments for training, design, and the communication of ideas.

Most visual languages to date have been two-dimensional. That is, they all have been displayed visually as though they were on a flat surface, or at best, on layered flat surfaces. Previously, the realm of three dimensions was confined to CAD modelers and scientific visualization applications, while visual languages were confined to flat representations. Computers were in most cases not powerful enough to render three-dimensional visual language constructs. With the advent of fast graphics-rendering hardware and devices supporting interaction with three-dimensional objects, three-dimensional visual languages become practical. Aside from being a natural extension to the use of a virtual environment, the third dimension (depth and perspective) can be used to provide a team of developers with a view of their entire project.

The Lingua Graphica prototype as designed encompasses three graphical programming techniques: programming by example, instance-based data flow simulation definition, and behavior definition.

Programming by example creates representative set of primitive operations for an agent or object to follow (placement of students, path definition, visual constraints and visual events, motion of items in response to state change).

Instance-based data flow simulation definition creates input and output ports for connecting agents or objects to define new system behavior.

Behavior definition defines methods for agent behavior and new agent classes using their three-dimensional visual representation.

Supervisor

The Supervisor is the agent responsible for controlling the state of the Training Studio as a whole. It arbitrates between the demands of different agents to change the collective Training Studio state. It also controls starting and stopping the various cooperating processes that form a Training Studio session, such as one or more Vista processes, agent frameworks, etc.

The Supervisor is the agent responsible for controlling the state of the Training Studio as a whole. It arbitrates between the demands of different networked agents to change the Training Studio state, and for agents in its own agent framework, it decides which agents will receive incoming messages.

To arbitrate Training Studio state change, the Supervisor keeps track of which agents are in control of the scene. If another requests a change, its request is put on hold until the current controlling agents notifies the Supervisor that it is finished with the scene. This mechanism is especially useful when more than one Training Assistant is being used, and one needs to set up the next instructional scene, while another needs the current state a while longer.

The Supervisor takes TScript messages coming into the Smalltalk Agent Framework, and decides where those messages are to be applied. Different kinds of Agents operating in the Training Studio have the need to capture events such as object selection during lesson development or instruction. The Supervisor fills this role, allowing agents to capture events normally sent to the specific object involved. For instance, if the developer is using a placement or scaling agent to move other agents, the placement agent requests selection events, and the Supervisor does not send them to the agent selected, but to the placement agent. When the developer is done with the placement agent, it notifies the Supervisor agent that it no longer needs these events, and further selection events will go directly to the agent selected.

Critical Technologies

There are a number of technologies developed or advanced under the ITDM contract which we feel contribute to an effective virtual environment for training. Foremost among these was the architecture approach using the virtual environment as a display server, and developing a network protocol that supports updating many such VE display servers. The philosophy behind this approach was to abstract the creation and use of distributed, multi-participant 3D spatial scenes in such a way as simplify instructional development, and yet still provide required flexibility.

Our choices in arriving at this networked Application Programmers Interface (API) stem from the principle of removing implementation detail while preserving the flexibility to compose a functional system from the building blocks in the API.

The Smalltalk agent class system provides a working example of the use of the network API to compose more complex systems. It uses the network API features for geometric primitives, selection events, and scene graph modification. We have shown it is possible to compose interfaces that allow creating, deleting and modifying objects, editing scene graphs composed of these objects, placing objects, and entering text while immersed.

The Smalltalk agent class system is one example which was used to test network composability, and from which other examples can be derived. At the USAF Technical Training Research Division, The RIDES system has also been used as another way to control instruction in a spatial setting using this ITDM network API, TScript. A Java-based system that uses the TScript network API is also eminently possible under exactly the same approach as the Smalltalk agent classes were built [Munro94].

Distributed VE Training Protocols

The distributed protocol we developed for the ITDM effort is called TScript. This protocol consists of groups of messages which can change visual scenes and convey changes in the distributed state of the Training Studio. It is not meant as a programming language, but rather as a common collection of commands (protocol) that can be used from inside of programming languages, or other tools or utilities connected to the communication bus [McCarthy93].

The categories of TScript messages are: object control, scene graph manipulation, event monitoring, and participant control. TScript supports creating 3D objects, constraining them, modifying them, and deleting them. Since the objects are part of a scene graph, TScript supports modifying the scene graph transformations, switches, and graph structure so that objects are transformed, visible, and modifiable in a very flexible manner over the networked communications bus. Participant Events can be registered with each Vista Viewer, and the participants viewing and interaction modes can be controlled using TScript. TScript supports participant scope on almost all messages dealing with participants, so that individuals can be affected specifically, or the whole group can be affected by the message.

Under the auspices of the ITDM contract, we have participated in the development of the VRML standard since June of 1994, extracting lessons learned from our own distributed virtual environment work, and informing the VRML community of them [Pesce95]. The VRML 1.0 scene format (Virtual Reality Modeling Language) is much like the HTML (Hyper Text Markup Language) format, but instead of describing documents, its describes 3D scenes. It is derived from Inventor, but differs from Inventor in that it supports live http links across the Internet. The VRML format is an analog to the HTML format used in World Wide Web documents, and uses the same http protocol used for HTML documents.

The implications of VRML capability for instructional authoring are good. Because the links are live, a group developing instruction using the training studio could deliver an initial training package to its users, and if user changes in the models were requested, the developing group could revise its models, put them on its WWW server, and know that all the releases of its software now have updated models incorporated in them. Other possibilities exist in collaborating with several groups to develop a complex spatial setting for instruction. Several groups could break up the task, and work on modeling selected components, linking them together in the complete scene, and where possible, use links to detailed models of standard items published by others on the Internet.

Distributed Interpolation

Distributed interpolation is the local update of a distributed object. In simulation applications, this update of values is most often applied to an object's position and orientation. Distributed interpolation is useful when one model of a simulation exists, and it is being viewed in several places (independent processes) in a distributed fashion. By distributing the interpolation, the interpolation can be started by one controlling simulation. Any updates can then happen locally in simulation systems until any correcting updates are sent by the simulation which owns the object, after it detects a mismatch between its high-fidelity values for the object and the interpolated values. A simple case of an interpolator is a rotator, wherein an object is caused to rotate about a center point at a given rate, until stopped. A more involved example is an object path, where way points and velocity are given, and the object moves until it completes its path, or is told to stop by a message.

Essentially, distributed interpolation provides support for what we termed physical dynamics in earlier ITDM reports, but acknowledges the distributed nature of the system.

The Distributed Interactive Simulation (DIS) dead-reckoning is a case of this that can be more generally applied. In the case of DIS, a description of an object's position, velocity, orientation, and change in orientation is sent over the network so that other systems may view the object. The other systems use the velocity and change in orientation to dead-reckon the object until the next update comes. The simulation that owns the object measures its accurate model of the object against the dead-reckoned model, and when there is a significant mismatch over a given threshhold, sends out a network update.

The Vista viewer supports orbital mechanics interpolators, dead-reckoned interpolation using the DIS message protocol, interpolators for orbital trajectories, and path interpolators which work on objects or persons. In addition, when a person is moving about free in a scene, updates are sent out on the communications bus for that participant, so that others in the session can perceive where the participant is. These values are sent out when the change in position or orientation is over a given threshhold.

Participant Models

Several enhancements were made to Vista functionality. The first of these was support for constraints on objects. Arbitrary, named transformation matrices (Performer DCS's) in Vista can be constrained for yaw, pitch and roll, as well as in the X, Y, and Z axis. The constraints can have an upper or lower limit, or both. They can also be made to preserve their same value, which is the form of object locking that can currently be set using TScript messages.

Functionality for the wider range of constraints has been tested. Any independent position or orientation value for an object can be constrained. The locking functionality per object was built in support of the sponsoring agency's immersed demonstrations where the training methodology demands that only certain objects can be moved by the student.

Through the ITDM work, we have arrived at a representative set of events that characterize a participant's interaction with the environment fairly well. New C++ classes for logging and monitoring events per participant while immersed were developed. These events are report events associated with the participant's viewing frustum, entry of objects into spherical, boxed, or cylinder regions, as well as intersection of object geometry with line segments defining arbitrary regions. By abstracting these events, domain experts do not have to become familiar with details of computational geometry in order to find out what is occurring in a spatial setting composed of one or more uncontrolled elements, most notably the students themselves.

Many devices are now available for use with virtual environments, and many more are likely to be introduced. Our treatment of devices largely removes the instructional developer from having to worry about which device was used to interact or view a given instructional setting. Selection events were abstracted to the object which was selected and the location on the object, with no mention of whether it was a CyberGlove, flat-screen mouse, or Ascension 3D mouse that the student used to select an item. This abstraction applies to the viewing systems as well. The instructor does not need to be concerned about the device used to view a setting, whether it is a FakeSpace boom, head-mount display, or normal computer monitor. They can develop the course in a spatial setting knowing the student has a view onto it and a way of interacting with it, and really focus on the domain itself.

Guiding the student's view onto simulations and other 3D examples is very important in the context of training methodologies in a spatial setting. We developed several viewing modes for controlling the views of individuals in the training studio. These modes are: viewing paths with focus objects; locking the view into a sphere around a focus object;locking the orientation of

a student to focus on an object while allowing free movement; and tethering a student to an object's motion while allowing freedom of view orientation.

Lessons Learned/Long Range Development Plan

This section discusses the technical results obtained during the course of the contract, as well as lessons learned. A conscious effort was made to be frank in terms of discussion, so as to have a positive impact on any future efforts.

Overall, we believe the work originally proposed for the ITDM program was too ambitious for the requested level of funding, which by the end of the two and a half year program was roughly $460k. The responsibility for this miscalculation of effort required for a new technical field lies with us, but the lesson learned is of importance to the funding agency. We believe that we delivered more technical results and software than is normally possible for the given level of funding, but upon examination of similar programs such as the Canadian Ship Handling Training program, Bowen Loftin's Hubble training system, Lockheed Martin's Simulation Based Design Phase I [Magee95, Loftin95, Sabatini94] for DARPA, and others, each of which was funded at more than 6 times the amount we worked with each year, we have determined that critical mass is above $1 million for each year of a VE program. At this level, enough funding is available for developers to focus on the project at hand and not timeshare with other projects. At or above this level, significant inter-company interest and support is generated, and resources such as computer hardware and software and program alliances within a large company become possible. Having discussed the lesson of appropriate funding for this type of technical effort, we also emphasize that the funds we did receive were greatly appreciated and put to effective use.

Our suggested long-range development plan for the technologies developed under ITDM is outlined in the Virtual Environments for Training proposal [Stiles94].

Virtual Environment System

In this report we have already discussed system capabilities, but one positive lesson learned was that by keeping the rendering system (Vista) separate from behavior/simulation allowed the TTRD to experiment with using other simulation systems (RIDES) to drive scene changes, object simulation, and instruction as well.

A number of tradeoffs in introducing a new technology for a 6.2 program versus devoting effort and resources in support of end-use evaluation came up. Early in the program we introduced work towards the 3D visual language Lingua Graphica, but return emphasis from the funding agency was minimal, and so more development resources from the finite amount available were devoted to end-use support and evaluation, where there was a great deal of emphasis from the funding agency. Support items such as a user's guide, detailed handling of bug reports, and frequent software updates were a result of this end-use evaluation. This is not to say that end-use evaluation was not important, but that given limited resources, it forced a choice to limit development of the 3D language. Nor is this to say that the 3D visual language was not important, or was somehow invalidated. We intend to revisit development of the 3D visual language, and pick up where we left off.

We believe we introduced the concept of editing a scene while immersed at a point too early for acceptance. It is possible to edit and save 3D scenes while immersed in the Training Studio virtual environment using the agent framework. This editing approach was not perceived as useful and was not adopted or emphasized during the evaluation process, in part because of previous reliance on using 2D layout and in part because of the raw prototype nature of the

system, in which the method of saving scenes changed and older scenes were not recoverable unless modified through editing by hand. More will be done with respect to immersed editing and reliable scene storing in the follow-on efforts.

Spatial Instructional Systems

A lesson we learned with respect to the underlying system supporting instruction was that the scripting language for the agents should have wide acceptance. We made a decision to use GNU Smalltalk, which is portable and has many good properties, but we should have used Java. The GNU Smalltalk system has not been updated by GNU since our decision, and in some ways is an orphan system. Smalltalk as a language is not orphaned, and is in many ways equivalent in expressive power to Java. Indeed, if one checks into Java's class structure, a very heavy borrowing from Smalltalk concepts is evident. However, Java was built with a conscious effort toward platform independence and network support, and because of its wide acceptance, will have more in terms of development support tools and development effort.

As mentioned in critical technologies, we believe the introduction of abstracting object creation, spatial layout, viewpoint controls, and user events in the manner which we did will have a positive impact for adapting the system to new instructional domains and software component tools. This aspect has allowed us fairly good progress in our follow-on efforts for VET.

Follow-on Application (VET)

We are in the process of applying the technology developed under the Intelligent Training Development Methodologies to the Virtual Environments for Training Focused Research Initiative sponsored by the Office of Naval Research.

This effort incorporates our work on Vista and networked update of virtual environments. The USC Information Sciences Institute (ISI) and USC Behavioral Technology Laboratories (BTL) are also contributing software components. BTL is developing V-RIDES, an extension of RIDES which will be part of the Training Studio system. V-RIDES controls object simulation for the purposes of training. ISI is developing STEVE, which uses the SOAR agent system and the JACK human animation system to display agents as team members or guides in spatial tasks in the Training Studio [Hill93]. The sponsoring agency for ITDM, the Technical Training Research Division, will be working on VET to evaluate the system effectiveness iteratively with the developing organizations.

The VET effort expands on the original ITDM idea of the Training Studio system to incorporate V-RIDES as an alternative for the Training Assistant, and to incorporate STEVE as a more advanced variant of the agents in the original agent framework. Because the Training Studio software, such as Vista, was designed to meet the requirement for incorporating disparate components, our initial work on VET has progressed fairly well along these lines without a major revision of the system.

Delivered Items

As part of summarizing the Intelligent Training Development Methodologies research effort, it is useful to itemize those items delivered as a result of the effort. Foremost among these was the alpha versions of the Training Studio software. By releasing versions of the software as new elements were developed, interaction with the funding agency to improve the final product was achieved, and in a more detailed and effective manner than with contract technical reviews alone. This section describes items delivered under contract, as outlined in the contract

deliverables document. It is not an exhaustive itemization, rather a summary of items delivered.

Monthly Status Reports (CDRL A001, A002)

Monthly status reports on contract technical progress were delivered officially in paper form by the Lockheed R&DD contracting office, and in an informal, convenient and timely manner direct by email to the Contract Office Technical Representative (COTR). These summary reports covered monthly work, instructions on the use of newly developed features, and explanations of the impact of new features as well as project status. Funds and Man-Hour Expenditure Reports were also included with each monthly status report.

Software Product: VR Tool Kit (CDRL A003)

The full trade name for the VR Tool Kit software is the Training Studio. Software was delivered, in several alpha release stages, that accomplished the orbital mechanics training demonstration, in a networked, real-time immersive VR setting. The details of design, performance, and operation of this software product are covered elsewhere in this report. A complete copy of all the project source code and executable software was delivered in December 1995.

Presentation Material (CDRL A004)

Several on-site reviews were conducted in Palo Alto, California. In addition to support for these reviews, the Contract Office Technical Representative (COTR) was provided with the Presentation materials for the reviews, either at the time of the review, or within 2 months of the review in the case of vu graphs of high-detail or color detail.

Final Report (CDRL A005)

This document is the final report. This report covers the technical items accomplished during the contract, summarizes the final system developed, recaps the contract deliverables, and assesses the impact of critical technologies developed, as well as indicating further directions for the software system.

Product Specification Report (CDRL A006)

This 28 page report was delivered in November of 1994. It describes the system as it should be delivered to the customer. It is a result of the requirements and system design and development efforts and outlines what the final product should do.

Requirements Report (CDRL A007)

The requirements report was developed in the first 3 months of the contract, as a prelude to the research and development effort. This 70 page document discussed technical capabilities required to meet instructional goals as part of a virtual environment system, and was used repeatedly during the course of the contract as a source of direction.

User's Guide (CDRL A007)

In conjunction with the VR Tool Kit software releases, a user's guide detailing usage of the system was provided. It describes the system built, discusses the functionality in the system

and its usage, and serves as a reference for the various distributed messages that are used in the system (TScript).

Informal Technical Information (CDRL A007)

During the course of the contract, an informal exchange was conducted by email between the Technical Training Research Division and the ITDM contract group. The email messages included, but were not limited in scope to: information and advice for purchasing and configuring hardware and software for updating a VR Lab at Brooks AFB , updates on current state of VR research outside the contracted research effort, and software release notes, instructions and clarifications.

Conclusions on Potential

The Training Studio was on the premise that the best way to ensure effective delivery of instruction in a networked virtual environment is to provide a structured metaphor and a set of tools to develop instructional material using the same environment where instruction will be delivered. It was also built from the start with the capability to incorporate new or revised components, and this has proven to be an asset.

With the Training Studio software, we have realized a software system in which it is possible to construct and deliver training scenarios which are delivered in an immersed, interactive setting. The proposed final goal of an orbital mechanics demonstration was achieved early in the course of contract and we went on to build a more general system. This more general system will see its fullest application in the Virtual Environments for Training program, where the generality of separating viewing/interaction from simulation will be used to integrate the RIDES and SOAR systems.

We see the Training Studio software in its final delivered form as an Internet-capable suite of software for developing and delivering immersive simulation-based training in a virtual environment. We see training content being developed at sites like the TTRD, essentially VE Training Centers, by domain experts and technical assistants, and then being fielded using the world-wide-web protocols to other Defense Department sites in a time-efficient and responsive manner. Changes to curriculum, simulations and 3D models can be distributed quickly. We feel the Training Studio going beyond what is available in current world-wide-web tools such as flat screen VRML browsers by supporting true immerseive, real-time interactions, and by supporting the kinds of instructional abstractions we have developed under ITDM.

References

[Grant91] Grant, F., McCarthy, L., Pontecorvo, M. & Stiles, R. *Training in virtual environments.* Conference on Intelligent Computer-Aided Training. Houston, TX: Johnson Space Center, November 1991.

[Hill93] Hill, R.W., Johnson, W. L., *Designing an intelligent tutoring system based on a reactive model of skill acquisition.* Proc. World Conf. of Artificial Intelligence in Education, pp. 273-281, 1993.

[King95] King, T. E., McDowell, P. L., *A networked virtual environment for shipboard training,* Masters Thesis March 1995, Naval Postgraduate School, Monterey CA 9343-5000. (see http://www-npsnet.cs.nps.navy.mil/npsnet/publications.html)

[Loftin95] Loftin, R. B., Kenney, P.J., *Training the hubble space telescope flight team,* IEEE Computer Graphics and Applications special issue on Virtual Reality, Rosenblum, J. R., Bryson, S., Feiner, S. K., editors, September 1995.

[Magee95] Magee, Lochlan E., Virtual reality simulator for training ship handling skills, Proceedings NATO Research Study Group 16: Adv. Technologies Applied to Training Design. Portsmouth, England, March 1995.

[McCarthy93] McCarthy, L., Stiles, R., Pontecorvo, M. & Grant, F. *Spatial considerations for instructional development in a virtual environment.* 1993 Conference on Intelligent Computer-Aided Training and Virtual Environment Technology. Houston, TX: NASA Johnson Space Center, May 1993.

[Munro94] Munro, A., Pizzini, Q., Towne, D. M., Wogulis, J. L., Coller, L. D., *Authoring procedural training by direct manipulation.* Working Paper WP94-3, Behavioral Technology Laboratories, USC, Redondo Beach CA., October 1994.

[Pesce95] Pesce, Mark, VRML, browsing and building cyberspace, Copyright 1995, New Riders Publishing, Indianapolis, IN.

[Rohlf94] Rohlf, J. & Helman, J., *IRIS Performer: a high performance multiprocessing toolkit for real-time 3D graphics.* Proceedings SIGGRAPH, Orlando FL, August 1994.

[Sabatini94] Sabatini, J. F., *Simulation-based design final report,* DARPA Contract MDA972-93-C-0028, Arlington, VA, October 1994.

[Stiles94] Stiles, R., Johnson, W. L., Munro, A., Virtual environments for training focused research initative proposal, Office of Naval Research for AFOSR BAA 94-06, August 1994.

[Stiles95] Stiles, R., McCarthy, L., Pontecorvo. *Training studio: a virtual environment for training.* 1995 Workshop on Simulation and Interaction in Virtual Environments. Iowa City, IW: ACM Press.

[Stiles93a] Stiles, R., McCarthy, L., Pontecorvo, M. & Grant, F. *Systems requirements report for intelligent training development methodologies.* Defense Technical Information Center, August 1993.

[Stiles93b] Stiles, R. & Milgram, D. *Advanced forms of stealth viewer control.* Proceedings of 9th Annual Distributed Interactive Simulations Workshop. Orlando, FL UCF-IST, September 1993.

[Stiles92] Stiles, R. & Pontecorvo, M. *Lingua Graphica: A Visual Language for Virtual Environments.* Proceedings of the IEEE International Workshop on Visual Languages. Seattle, WA: IEEE Press, September 1992.

[Tate95] Tate, D. L., Sibert, L. Wiliams, F. W., King, T. E., Hewitt, D. H. Virtual enviroment firefighting, ship familiarization feasibility tests aboard the ex-USS Shadwell, Naval Research Lab Technical Report 6180/0672A, Washington, DC. 1995.

Appendix A: Communications Bus & Tscript

The Training Studio is a virtual environment where several people can meet for training. To support this, several Vista Viewers and agents can be assembled together on the network, and kept synchronized together using distributed protocols across a communications bus. The distributed protocols is a set of messages which change the world model of the Training Studio. Two main forms of distributed communications are supported in the Training Studio. These are our own object-oriented, training specific protocol called TScript, and the ISO standard Distributed Interactive Simulation (DIS) protocol. We developed TScript to address specific training-oriented needs in a networked virtual environment, and we adopted the widely accepted DIS protocol to open up the Training Studio to the wide array of vehicle-oriented simulations that are available for use under this standard.

The mechanism by which the processes in the system communicate information is called the communications bus, a software abstraction of a computer bus. The distributed protocol by which the processes communicate is called TScript, and the transport mechanism for this distributed protocol is ToolTalk, originally developed by Sun Microsystems. ToolTalk was chosen for the transport system because it allows processes to register interest in messages, much like a hardware component selecting certain messages addressed to it from a large set of messages on a computer bus.

Arch. Abstraction	Element	Software Element
Communications Bus	Participants	Vista(s) and Agents
	World Model	Scene Graph w/objects
	Distributed Protocol	TScript (Ontology)
	Transport Mechanism	ToolTalk
		RPC
		TCP/IP

This TScript protocol consists of groups of messages which can change visual scenes and convey changes in the distributed state of the simulation. It is not meant as a programming language, but rather as a common collection of commands (protocol) that can be used from inside of programming languages, or other tools or utilities connected to the communication bus.

The categories of TScript messages are: object control, scene graph manipulation, event monitoring, and participant control. TScript supports creating 3D objects, constraining them, modifying them, and deleting them, in one or more Vista Viewers. Information about the state of these objects is also conveyed using the TScript protocol to any other processes on the communication bus.

Since objects created by TScript are part of a scene graph, TScript supports modifying the scene graph transformations, switches, and graph structure so that objects are transformed, visible, and modifiable in a very flexible manner over the networked communications bus.

TScript supports modifying the properties of objects, such as constraints on objects. Arbitrary, named transformation matrices in Vista scene graph can be constrained for yaw, pitch and roll, as well as in the X, Y, and Z axis. The constraints can have an upper or lower limit, or both. They can also be made to preserve the same value, which is a form of object locking. Functionality for the wider range of constraints has been tested. Any independent position or orientation value for an object can be constrained. The locking functionality per object was built in support of immersed training applications where the training methodology demands that only certain objects can be moved by the student.

We have arrived at a representative set of events that characterize a participant's interaction with the environment fairly well. TScript messages for logging and monitoring events per participant while immersed have been developed. These events are reported as TScript events associated with the participant's viewing frustum, entry of objects into spherical, boxed, or cylinder regions, as well as intersection of object geometry with line segments defining arbitrary regions. By abstracting these events, domain experts do not have to become familiar with details of computational geometry in order to find out what is occurring in a spatial setting composed of one or more uncontrolled elements, most notably the students themselves.

Using TScript messages, participant events can be registered with each Vista Viewer, and the participant's viewing and interaction modes can be controlled using TScript. TScript supports participant scope on almost all messages dealing with participants, so that individuals can be affected specifically, or a whole group can be affected by the message.

Many devices are now available for use with virtual environments, and many more are likely to be introduced. Our treatment of devices largely removes the instructional developer from having to worry about which device was used to interact or view a given instructional setting. Selection events in TScript were abstracted to the object which was selected and the location on the object, with no mention of whether it was a CyberGlove, flat-screen mouse, or Ascension 3D mouse used by the student to select an item. This abstraction applies to the viewing systems as well, the instructor does not need to be concerned about the device used to view a setting, whether it is a FakeSpace boom, head-mount display, or normal computer monitor. They can develop the course in a spatial setting knowing the student has a view onto it and a way of interacting with it, and really focus on the domain itself.

Guiding the student's view onto simulations and other 3D examples is very important in the context of training methodologies in a spatial setting. We developed several viewing modes for controlling the views of individuals in the training studio. These modes are: viewing paths with focus objects; locking the view into a sphere around a possibly moving focus object; locking the orientation of a student to focus on an object while allowing free movement; and tethering a student to an object's motion while allowing freedom of view orientation.

The Shared World Model

The shared world model is an important topic when discussing the use of the communications bus. The TScript protocol is a set of messages which modify a scene graph or a participant's interaction with a scene graph. The scene graph is a directed acyclic graph used to represent a scene. There is a root node which anchors the scene, called vrScene. As children of this node there can be transformation matrices, switch nodes, or geometry nodes.

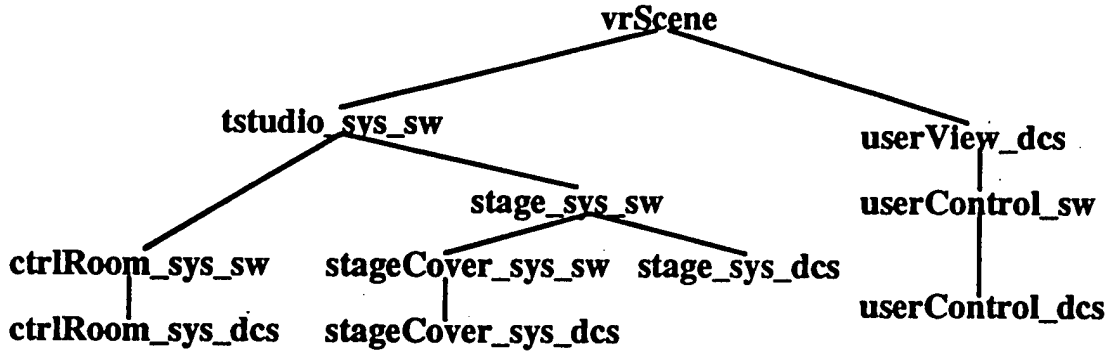
Vista traverses the scene graph from the root vrScene to the leaf nodes every time it draws a picture on the screen (actually Performer underlying does, but the scene graph is a very common concept in 3D graphics). Starting from the scene root, it traverses each child from first to last, depth first. As it traverses the scene graph it multiplies each matrix it encounters against a previous composite 4x4 matrix, and when it arrives at geometry, it uses the resulting composite matrix to multiply the vertices of the geometry. This is how you make things move in Vista, you change the values for position, orientation or scale of a transformation matrix in the scene graph, and any children below it are modified accordingly.

When Vista traverses the scene graph, it pays attention to switches as well. If it encounters a switch that is turned off, it goes no further down that branch of the tree, and so no part of that tree will be drawn. If the switch is on, it proceeds, and will eventually draw it.

Switches and transformation matrix nodes (dcs, for dynamic coordinate system) can have any number of child nodes, which may be other switch, dcs nodes, or geometry nodes. Geometry nodes may not have children.

Following is a figure representing the scene graph that Vista starts out with every time. At the root is the named node vrScene, and as one child is the training studio (tstudio_sys_sw) and as another child is the userView_dcs. The userView_dcs is updated each time Vista draws a scene to reflect the world coordinates of the user (participant) who is using vista. Interface controls that move with the user may be attached to the tree below userView_dcs, namely at the named node userControl_dcs. TScript messages to set the position etc. for userControl_dcs will move the interface object relative to the viewer, not the world.

The scene graph under tstudio_sys_sw consists of the control room and the stage, and the stage has a sub-tree representing the stage cover (roof). Objects meant for use by the instructor should always be attached to the named node ctrlRoom_sys_dcs, while object that are intended for the student participants should always be attached to the stage_sys_dcs. This is a convention in the world model that is quite useful, and should be adhered to. It is also possible to add nodes to the top-level scene itself, but it is not recommended.



Another part of the shared world model used in the Training Studio are conventions for measurement.

- Distances/Rates/Accell etc. are all in meters
- Orientation values for any angles values are always in degrees
- Colors are expressed with red, green, blue, and alpha components, and range from (0,1) as floating point numbers.
- Object priority is expressed in the range (0,1) as a floating point number.
- When using scale for objects, a scale of 1.0 will leave it at its current size. If an object is a 2x2x2 cube, and it is scaled by 10 it will have the dimensions 20x20x20, meaning it will now be 20 meters in each of dimensions.

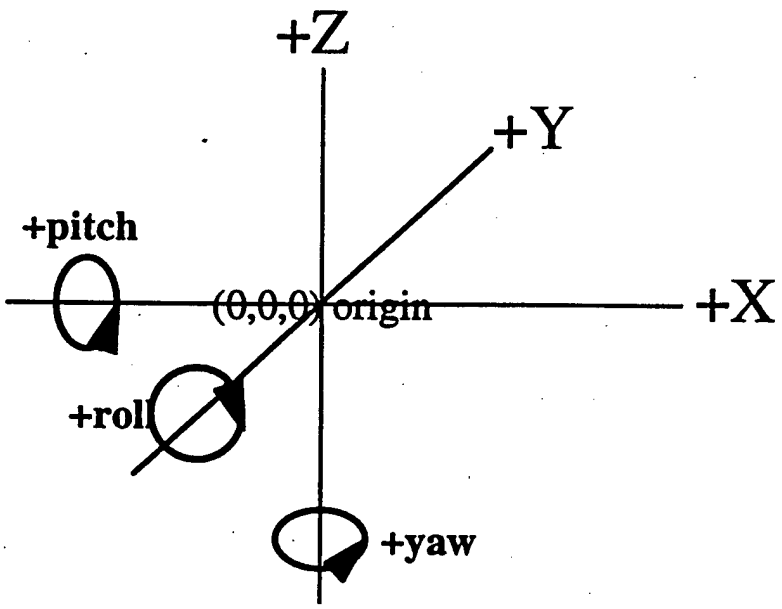
Another convention in place for the Training Studio is that it operates as a Cartesian space where angles are expressed using the right-hand rule.

When Vista is initialized, the Training Studio stage is always placed so that the middle of the floor is the origin (0,0,0) in this Cartesian space. Giving an object coordinates of 0,0,2 for its translation means that the object will appear in the center of the floor, with the object's

center 2 meters up from the floor. By convention the viewer always starts his session in Vista looking into the stage so that he can see the origin, and +X in world coordinates is to his right, +Y in world coordinates results in positions further into the screen, and +Z in world coordinates results in positions above the origin. Of course, minus values along these axis are the opposite. As the viewer moves around, the world coordinate system does not change. The same conventions apply for local coordinate systems, i.e. any and all nodes below a given transformation matrix (dcs).

Now for an explanation of the orientation conventions. There are three angles in every transformation matrix that represent an object's orientation, yaw, pitch, and roll. These are always in degrees. If your process works with angles in radians, be sure to convert them to degrees before sending them out in any TScript message.

Each angle is applied to an object in order. First the Yaw is applied to an object, then the pitch, and finally the roll. A positive yaw value is applied on the Z axis, and will turn the object counter-clockwise. A positive pitch is applied about the X axis, and will tip the front of an object upwards. A positive roll is applied about the Y axis, and will roll the object clockwise as you look down the Y axis. The front of any object is always along its Y-axis in its own local coordinate frame.



ToolTalk

ToolTalk is a TCP/IP and RPC, connection-oriented messaging system developed by Sun Microsystems and distributed by Silicon Graphics as well. Its purpose is to support messaging between software tools. In our use of ToolTalk, there is a central server (ttsession) which the components connect to. Each component registers interest in a set of messages, and when this message is sent to the server, all the clients that have registered interest receive a copy of the message.

One may think of the ttsession process as the actual incarnation of the communications bus abstraction. Each vista viewer and any other processes wishing to share the same world must connect to the same ttsession. The general user does not need to be familiar with ToolTalk, since it is just a transport mechanism, and not a separate language. They do need to know that ToolTalk is being used to check if their system has ToolTalk, and to check if the ttsession message router is up and running when troubleshooting.

TScript Protocol

The main distributed protocol used in the Training Studio framework is called TScript. The name TScript means *Training Script*. This protocol consists of groups of messages which can change visual scenes and convey changes in the distributed state of the Training Studio. It is not meant as a programming language, but rather as a common collection of commands (protocol) that can be used from inside of programming languages or utilities.

Object Creation

Different forms of simulation and graphic objects can be created for use and update inside of Vista. The Vista Viewer and Smalltalk framework maintain class definitions which represent most of the graphic objects. In Smalltalk, generating these TScript messages to create graphic objects in any connected Vista Viewers entails creating a new instance of a class and then initializing it to broadcast the appropriate TScript messages.

When a Graphic Object (GO) is built, Vista must attach it to part of the scene graph, and so the name of a parent node in the scene graph must be specified. When a TScript message for creating an object arrives to a Vista Viewer, it creates a switch node and a dcs node for the object, and gives them names by convention. So if you sent a vrBldGO message with stage_sys_dcs as the parent, and named the new object cube99, the stage_sys_dcs node would have a new child node called cube99_grf_sw, which by default is switched on, and this node will have a child node named cube99_grf_dcs, with the identity matrix as its default (effectively does no translation, rotation or scaling). When you add cube geometry to the graphic object named cube99 using vrAddToGO, it puts a set of colored vertices as a child of the cube99_grf_dcs node.

The convention is that all switch nodes created as part of object creation have *_grf_sw attached to their name, and all dcs node created this way have *_grf_dcs attached to their name. TScript allows the user to make single switch nodes and dcs nodes as part of the scene graph, and this can be quite useful for complex structures. By convention you should append *_sys_sw or *_sys_dcs for their names as appropriate.

1) All objects created must specify a parent
2) All arguments for TScript messages must be given
3) All objects will be created with an associated switch and dcs node

vrAddSimObj <sim-type>
Where the parameters are specific to the simulation type. Will use default model specified in any previous TScript message. There are currently only two such types of simulation object supported directly in Vista - other types, and future types should be supported in the agent framework.

satellite <name> <major> <eccentricity> <inclination> <ascension> <perigee> <red> <green> <blue> <alpha>

Accepted by Vista Viewer to build a satellite using 5 orbital element parameters and a color specification for the path and disks. The satellite scene graph components can be referred by name as <name>_sys_sw and <name>_sys_dcs, The disk and path associated with the satellite orbit are named as <name>_disk_sw, <name>_disk_dcs, <name>_path_sw, and <name>_path_dcs.

constelem <name>
<major> <eccentricity> <inclination> <ascension> <perigee> <true-anomaly> <red> <green> <blue> <alpha>
Similar to satellite sim type, but expects argument for true anomaly as well, so that constellations of satellites, each in correct phase with the other, can be built. Simulation should be turned off until all constellation elements of a constellation have been defined, then turned back on.

Various types of visual objects can be created and monitored in the Training Studio framework. The vrBldGO message builds a graphic object container, which you then add geometric shapes to. There is also a file-based variant which can load geometric files.

vrBldFGO <name> <parent> <model-file> <X> <Y> <Z>
Build container, add file-based geometric model to it. This message is handled by Vista. The model-file can be Flight, Inventor, Designer Workbench, or other format files that Performer can load.

vrFileObj <model-file-name>
Convenience message handled by the agent framework. This will cause the Profiler to make a representation of the file object, and send a vrBldFGO to all the Vista Viewers. The file model will load into the studio stage at the point 0,0,0 with all zero orientations. It will be named <model-file-name>1_grf_dcs.

vrSceneSave <scene-name>
Accepted by the agent framework, in order to save scenes. The agent framework will iterate through all graphic objects it has a representation for, and save them to a scene file. Information on scale, translation, rotation, switch settings, color, parent, etc will be saved, and can be loaded using *vrSceneLoad*. The file saved has the extension *.scene. Since this can result in saving large scenes, which take long to load, it is recommended that scene controls in the control workspace are used to select a small set of objects to be saved in a scene.

vrSceneLoad <scene-name>
Accepted by the agent framework, this message will cause it to look for a scene file of the same name (don't add the *.scene extension) and cause that scene to be loaded in the Vista Viewer. This load will work for scenes saved using the *vrSceneSave* message, or saved using the control panel.

vrBldGO <name> <parent> <priority> <X> <Y> <Z> <yaw> <pitch> <roll> <scale> <red> <green> <blue> <alpha>
The Vista Viewer handles this message. As a result of the message, Vista will build a graphic object container. You can then add various graphic shapes to this container.

All geometry, such as boxes, lines, etc. that are added to the graphic object container inherit its setting for color and transparency, as well as its overall coordinates.

The vrAddToGO message adds a piece of geometry to a given graphic object container. In this way, a complex, additive shape for a graphic object can be specified using the simpler geometries. The dimension parameters are always in meters.

vrAddToGO *<specifier>*
Where *<specifier>* can be any one of the following:

cone <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<sides> <base-radius> <height>
Create cone of varying height and radius

cyl <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<sides> <radius> <height>
Create cylinder of varying height and radius

box <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<height> <width> <depth>
Create box with different height, width, and depth

cf <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<sides> <base-radius> <top-radius> <height>
Create irregular cylinder, which has different radius values for each end

disk <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<sides> <radius>
Create flat disk with a given radius.

circle <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<sides> <radius> <line width>
Create circle formed by line, with given radius

circleD <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<num-points> <radius> <line width>
Create circle made of dashed lines

rect <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<width> <height>
Create flat rectangle

tri <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<X1> <Y1> <X2> <Y2> <X3> <Y3>
Create flat triangle, possibly irregular, using three points.

rTri <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<adjacent-length> <opp-length>
Create flat right triangle by giving opposite and adjacent sides

line <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<X1> <Y1> <Z1> <X1> <Y2> <Z2>
Create one line. Not dynamically updated or associated with other objects.

point <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<X> <Y> <Z>
Create point

quad <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<X1> <Y1> <X2> <Y2> <X3> <Y3> <X4> <Y4>
Create quadrilateral polygon - in the plane

squad <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<a> <b> <d> <c> <n1> <n2>
<u_min> <u_max> <u_step>
<v_min> <v_max> <v_step>
Create superquadric shape - based on forms of ellipses - includes spheres, eggs and toruses

sq <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<size>
Create square plane

wedge <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<num-points> <angle> <sides>
Create wedge

arc <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<npts> <angle> <radius> <line width>
Create arc, with rounded edge line

arcD <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<npts> <angle> <radius> <line width>
Create dashed arc with dashed edge. The argument npts gives the number of points used from 0.0 to your specified angle. The more points, the smoother its rounded edge appears.

text3d <name> <X> <Y> <Z> <yaw> <pitch> <roll> <complexity> <scale> <textstring...>
Based on Inventor 3D text, builds a text string in 3D so that it can be immersed in the scene. The complexity inicates how smooth/blocky the letters are and ranges from 0.0 to 1.0. A complexity of 0.3 is suggested to reduce polygon load in rendering scenes. The scale must be a positive number, and indicates the scale of the text. The final arguments are the text string itself.

imagescreen <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<rgb-imagefile> <screenwidth>
Build a rectangular screen of a given width in meters. The height is determined by the aspect ratio of the image itself to the width you provide so that images appear undistorted. The image shows on both sides of the screen, and the depth of the screen is a small ratio to the given width. The image should be an SGI rgb format image file. SGI Utilities such as *fromgif, frompppm,* and *fromsun* exist to get your favorite image into the rgb format.

sphere <name> <X> <Y> <Z> <yaw> <pitch> <roll> <radius> <complexity>
Buiild polygonal sphere.

wSphere <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<npts> <num-vert> <num-horiz>
<radius> <vert-width> <horiz-width>
Build wire sphere, such as lat/lon sphere used on earth

vrLink <from-dcs-name> <to-dcs-name>

Builds a dynamic link (line) from one dcs (dynamic coordinate system) to another. This link is parented under the from-dcs portion of the scene graph, so it will obey switches above it in the scene graph. Whenever either dcs moves, the link is updated. This is most useful when showing associations between objects in the virtual world (currenlty the link is always red, soon arguments will be available for rgb to set unique colors, for line width).

Object Deletion

Deleting an object also causes any swich or dcs nodes that were created with it to be removed unless they have other children not associated with the node named for deletion.

vrDelGO <obj-name>
Delete a named graphic object from the scene graph

vrDelSimObj <sim-type> <sim-obj-name>
Delete a named simulation object and all its attendant structures. Currently works for satellite simulation objects.

vrDelink <from-dcs-name> <to-dcs-name>
A dynamic link is specified by it from-to association. By sending a vrDelink message the the same dcs names as an existing link, you can force it to be deleted (un-linked).

Object Modification

Object transformation is the movement, rotation, and scaling of visual objects in a three-dimensional setting. We also include changes to parameters of the object here, such as color or transparency, complexity, or dimensions.

vrTranslateGO <obj-name> <x> <y> <z>
Translate an object to a given point. Translation is local to frame of reference, and not in world coordinates.

vrRotateGO <obj-name> <yaw> <pitch> <roll>
Rotate named object. Angles are in degrees.

vrScaleGO <obj-name> <scale-num>
Scale named object. Range is 0.0,100.0

vrDCSData <participant> <obj-name>
This message tells the Vista Viewer tor a given participant to find the named DCS and report back information on its position and orientation and scale. It sends out the TScript message vrcDCSData in response to this query.

vrcDCSData <obj-name> <x> <y> <z> <yaw> <pitch> <roll> <scale>
Result returned after a vrDCSData query. This gives information about position and orientation for a given named node's transformation matrix (dcs). So if your process sends a vrDCSData Tscript message, it should also register interest for the return vrcDCSData message.

vrBSphereData <participant> <node-name>
Accepted by the Vista Viewer (Performer 2.0) this message requests bounding sphere information for a named node in the scene graph associated with a given participant.

vrcBSphereData <participant> <node-name> <world-x> <world-y <world-z> <radius>
Return result from a vrBSphereData query, which has the world position, followed by the radius in meters of a bounding sphere which encloses all the scene graph elements below the named node.

vrBBoxData <participant> <node-name>
Accepted by the Vista Viewer (Performer 2.0) this message requests bounding box information for a named node in the scene graph associated with a given participant.

vrcBBoxData <participant> <node-name> <world-x> <world-y <world-z>
<min-x> <min-y> <min-z> <max-x>> <max-y> <max-z>
Return result from a vrBBoxData query, which has world position, followed by the minimum xyz and maximum xyz defining a bounding box which contains all the scene graph elements below the named node.

vrLock <dcs-name> <toggle-val-0-or-1>
Lock an object into place, so that it cannot be transformed or oriented differently

vrRollGO <obj-name> <roll-threshold>
Set threshold for roll whenever this object is traveling on a path. The roll threshold is in degrees, and if it is given a 0.0 argument, the object will not roll (bank) at all as it turns from one path segment to another.

vrConstrain* <dcs-name>
Set the constraints for a named transformation matrix (DCS). These can be constrained to an upper or lower limit, for any axis or orientation.

vrAttachPathGO <obj-name> <path-name>
Takes path argument, and stores this with graphic object so that the object can travel along this path once started.

vrStartPathGO <obj-name>
Similar to vrStartPA, this message tells Vista to start the graphic object along a path, if one has been previously attached to the graphic object.

vrStopPathGO <obj-name>
Similar to vrStopPA, this message stops an object if it is traveling along a path.

Event Monitoring

Tscript messages are used to report the occurrance of events as well as for registering them. Whenever an event involves an object or participant, the specific name of the participant or object is included in the message. Such event monitoring is necessary and useful in a setting where the people (participants) may do almost anything, and where disparate simulations are coming together to update a world model.

vrPick <object-name> <participant> <local-x> <local-y> <local-z>
Report pick event in Vista, pick point is local to the object picked. A vrPick message is sent out from a participant's vista viewer is output whenever the user selects a named object.

vrPOV <participant> <x> <y> <z> <yaw> <pitch> <roll>
Report point of view (POV) of participant XYZ in meters, orientation in degrees. Output from the Vista viewer periodically, according to threshold settings for change in position and orientation.

vrEvent <participant> <event-name> <event-type> <type-args>
Register an event for a given participant.

Event types can be the following for the vrEvent message can be the following:

sphere <dcs-name> <radius>
events are reported when other dcs center points are detected to be inside of the given sphere
spherecol <dcs-name>
events are reported when the sphere of another dcs is detected to be intersecting with this sphere
frustum participant
events are reported when the bounding spheres of other nodes in the scope are detected to be inside the participant's viewing frustum (Performer 2.0 version requires participant token)
frustum <dcs-name> <near> <far> <vertical-fov> <horizontal-fov>
events are reported for the defined frustum when the spheres of other nodes are detected to be inside this defined frustum (Performer 2.0 version only)
box*
segments*
cylinder* <dcs-name> <axis-vec> <half-length>

The scope for each vrEvent is modified using the vrEventScope message below.

vrEventScope <participant> <event-name> <duration> <scope-objs>
Set the scope for a named event, where duration can be the tokens *single, continuous, none, delete,* or a number indicating seconds since receipt of message for scope. The scope-objs can be any named objects in the environment, such as other participants, or graphic objects.

vrEventNotice <participant> <event-name> <scope-name> <e-type> <e-specs>
Events that are logged for a participant can result in a returned event notice. This gives the name of the registered event, and specifics for that event type describing the event. Output from the Vista Viewer whenever the conditions for a given event are met.

Participant Control

Participants in the Training Studio must at times be guided to assure effective instruction. The TScript primitives which follow help in the task of participant guidance. Note: in TScript messages which take a participant argument, this argument is currently their user name that they have logged in with. The special participant *all* can be used as a wild-card to send messages to all the participants at the same time.

vrPartMoveRel <participant> <dcs-name> <radius>
Move immersed participant near a transformation matrix <dcs-name> based on their current orientation - their view is moved so that the object is in the center of their view. Related to the spherical move.

vrPartSphereMove <participant> <object-dcs-name> <radius>
Tether an immersed participant to a named transformation matrix <dcs-name> a given radius away (in meters, scaled according to the tethered object). Effect is that immersed person can look up to see under object, look down to see top of object, and in general move their view in a sphere around an arbitrary moving or stationary object.

vrPartTrack <participant> <object-dcs-name>

Accepted by the Vista Viewer for a given participant. Causes orientation of the viewer to continuously change in order to follow a focus object. The participant can move around, but they stay focused onto the object. This viewing mode is like the focus for a path, but the user is free to change their position. (replaces vrLookAt in earlier versions).

vrPartPosition <participant> <world-x> <world-y> <world-z>
Accepted by the Vista Viewer, this sets a given participant's position in world coordinates. Replaces vrMoveTo, which doesn't differentiate between participants.

vrTether* <participant> <object-dcs-name>
Accepted by the Vista Viewer to tether a participant to a given named object. As object moves, so does the participant, though they can move relative to the tethered object.

vrRayLength <participant> <ray-len-meters>
Sent out and accepted by the Vista Viewer, this information indicates a participants ray length, and is used to update the ray representations in other participants viewers.

vrNearClip <participant> <far-clip-dist>
Set the participant's near clipping distance, in meters.

vrFarClip <participant> <far-clip-dist>
Set the participant's far clipping distance, in meters.

vrFov <participant> <horizontal-deg> <vertical-deg>
Set the participant's horizontal and vertical field of view, measured in degrees..

vrIpd <participant> <ipd-val-meters>
Accepted by the Vista Viewer, this sets the inter-pupillary distance for a given participant. It is in meters, so meaningful numbers should be 0.05, etc. for 5 centimeters. This value is used to separate the software channels in Performer by the given amount, so that views have a slightly different perspective, such as your eyes experience.

vrSkyModel <participant> <sky-model-number>
Set a given participant's sky model. Currently in range 0,5, where 0 is to clear to black such as is used for the orbital mechanics demonstration's black sky.

vrText <participant> <parent> <textname> <style> <size> <red> <green> <blue> <msgstring...>
Accepted by the Vista Viewer (Performer 2.0), this message creates or modifies the named text node in the 3D scene. The text displayed can have a style attribute of flat (2D polygons) extruded (3D) or textured (as an image).

vrSetText <participant> <textname> <msgstring...>
Accepted by the Vista Viewer (Performer 2.0), this message modifies the text string for already existing text in the 3D scene (created previouusly using vrText).

vrScreenText <participant> <text-bin-num> <text-string...>
Accepted by the Vista Viewer to display overlay text onto a given participant's view. There are currently 5 text bins. Bin 2 is reserved for displaying the mode that the participant is in, in the upper left corner. The screen text resulting from this message will always be in the middle of the screen, and starting from the left side.

vrScreenTextRel <participant> <text-bin-num> <rel-x> <rel-y> <text-string...>

Accepted by the Vista Viewer to display text relative to the screen. Similar to vrScreenText, except rel-x and rel-y arguments specify where on the screen the text appears. The origin is the lower left, and the range is (0.0,1.0). e.g. sending rel-x = 0.1 and rel-y = 0.5 will place your text on the left side, near the middle of the screen vertically.

vrScreenTextClear <participant>
Accepted by the Vista Viewer, this message will cause it to clear the screen of all text. If you desire just one text-bin to be cleared, send it a vrScreenText message for that text bin with an empty screen.

vrSetThreshold <participant> <thresh-type> <threshhold-value>
Accepted by the Vista Viewer, this message will set a type of threshhold for a given participant. The types can at present be: *translate* or *rotate*, and again these are in meters and degrees, respectively.

vrBldPA <path-name>
Build a path container using given name

vrAddToPA <path-seg-type> <path-name> <path-seg-parameters>
Add path segment to a named path, with specific path segment parameters

Path segment types and their respective parameters follow:

speed <path-name> <desired-speed> <rate>
Set the speed of travel for all subsequent path segments. Overridden by any following speed segments.

delay <path-name> <seconds-delay>
Set a delay segment into path. Object or participant will wait at endpoint of previous segment until this delay is expired.

fillet <path-name> <turn-radius>
Specify smooth turn between previous path line segment and next path line segment. Turn radius argument is given in meters.

arc <path-name> <pivot-x> <pivot-y> <pivot-z> <radius> <angle1> <angle2>
Specify explicity arc around a given pivot point in world coordinates. The radius is in degrees. The first angle modifies the object's yaw, the send angle modifies it's roll as it progresses through the arc.

line <path-name> <x1> <y1> <z1> <x2> <y2> <z2>
Specify path segment where object or participant is oriented to look down the segment as the travel it. The arguments are world coordinates for start and finish points of the line.

lineOO <object1-name> <object2-name> <x1> <y1> <z1> <x2> <y2> <z2>
Specify path segment that points object or participant orientation at object1 to start with, and finishes pointing at object2. The other arguments are the world coordinates for a start and end point for the path segment.

lineDD* *not enabled*

lineDO* *not enabled*

vrSelectPA <path-name>

Select path name for further use (soon to be phased out - redundant)

vrStartPA <participant>
Handled by the Vista Viewer for a given participant. Start previously selected path (soon to take path-name argument and obsolet vrSelectPA)

vrStopPA <participant>
This message is handled by the Vista Viewer for a given participant. Stop any path currently in progress.

vrNewText* <text-name> <double-quoted text>
Send text for Vista Viewers to display

vrMakeLabel* <object-dcs-name> <double-quoted text>
Send text to Vista Viewer to have it display moving text label

vrMoveTo <x> <y> <z>
Move participant to a given place. (Soon to take participant name as argument)

vrLookAt <obj-name>
Set participant's orientation (without moving them) to look at a given object. (Soon to take participant name as argument)

Scene Control

vrSwitchGO <obj-name> <switch-num>
Set switch to 0 to turn off objects in scene, 1 to turn on

vrAddSwitch <parent-name> <switch-name>
Add a switch node to the scene graph - useful if need to add objects below

vrAddDCS <parent-name> <dcs-matrix-name>
Add a transformation matrix to scene graph

vrAddGroup <parent-name> <group-node-name>
Add a group node to scene graph - used to collect children together

vrFOV <participant> <field-of-view-value>
Set the field of view for a participant range in Derek 0,90

vrReparent <new-parent-name> <node-name>
Set a named node in the scene graph to have a new parent

vrLodScale <new-lod-scale>
Set the scale for level of detail transitions, range 0.0,100.0

vrFindParent <named-node>
Vista viewer will respond to this query with a message giving parent of named node in the scene graph. Useful for traveling up the scene graph from a picked object to affect a wider collection of objects.

vrcParent <parent-name>
Output by the Vista Viewer in response to a *vrFindParent* query.

Session Control

vrAllCall
Requests all components connected to communication bus to identify themselves by responding with a vrComponent message

vrComponent <component-name>
Response to all call request

vrStop <component-name>
Stop execution for named component, if *all* is used, all will respond

vrStopSave <component-name>
Stop after saving state if capable of saving state

vrDebugMsgs <toggle-val-1-or-0>
Accepted by the Vista Viewer, this message will toggle the printing of all incoming TScript messages to standard output. Very useful if you are building a utility that sends TScript commands to Vista, and you need to be certain they are formed correctly (according to this user guide section).

vrToggleSim
Accepted by the Vista Viewer , globally toggle on or off the whole simulation in Vista

vrSimRate <seconds-multiplier>
Accepted by Vista to set the rate for the global simulation.

vrClockNew <clock-name>
Create a new simulation rate clock

vrClockRate <clock-name> <rate>
Set the rate of a named simulation clock

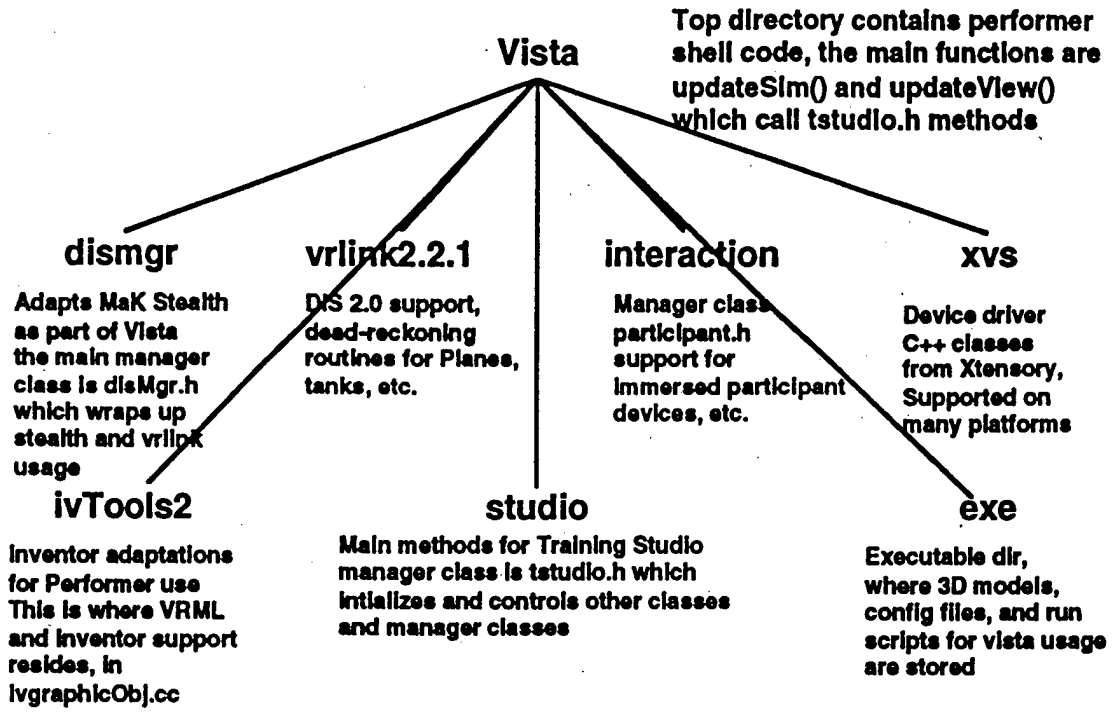
vrClockToggle <clock-name> <tog-val-0-or-1>
Toggle on or off an individual, named simulation clock.

vrClockSet <clock-name> <new-time>
Set the time value for a named simulation clock

Appendix B: Vista Software Internals

Source code for all the ITDM project software was delivered to the sponsoring agency. To aid in understanding the most complex software component, the Vista Viewer, this appendix contains brief discussions of the Vista Software Internals.

The following figure is a roadmap to the Vista source code directories, so that people working to understand Vista internals know where to look for particular functionality.



Graphic Objects

The C++ class for graphic objects is in graphicObj.cc, in the studio directory.

Each graphic object is made with
- a material (coloring, shininess, emmisveness)
- a switch *_grf_sw
- a dcs *_grf_dcs (transformation matrix)

Each graphic object is maintained using an open hash table called objTable in the tstudio.h class. When a graphic object is constructed, it enters itself in this table for other reference, and when it is deleted, it removes itself.

When messages to modify graphic objects come into Vista over the communications bus, the name is used in the hash table to get areal pointer to the graphic object.

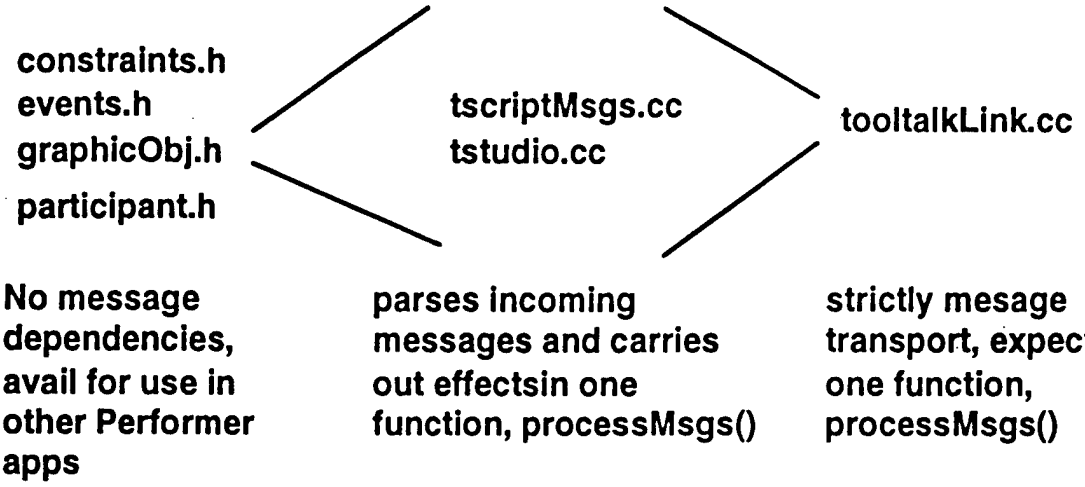
Tscript Messaging

All transport is handled in tooltalkLink.cc (see Vista Messaing Figure)
- mttRegister() to register interest in receiving a given type of message
- mttSendNotice() to send out messages on the communications bus

All message handling is done in tscriptMsgs.cc
- processMsgs(char *op, char *args)
- called by code in tooltalkLink.cc
- in turn calls specific functions in tscriptMsgs.cc for each type of Tscript message

Tscript Functions
- handle creating/deleting graphic objects
- handle displaying screen text

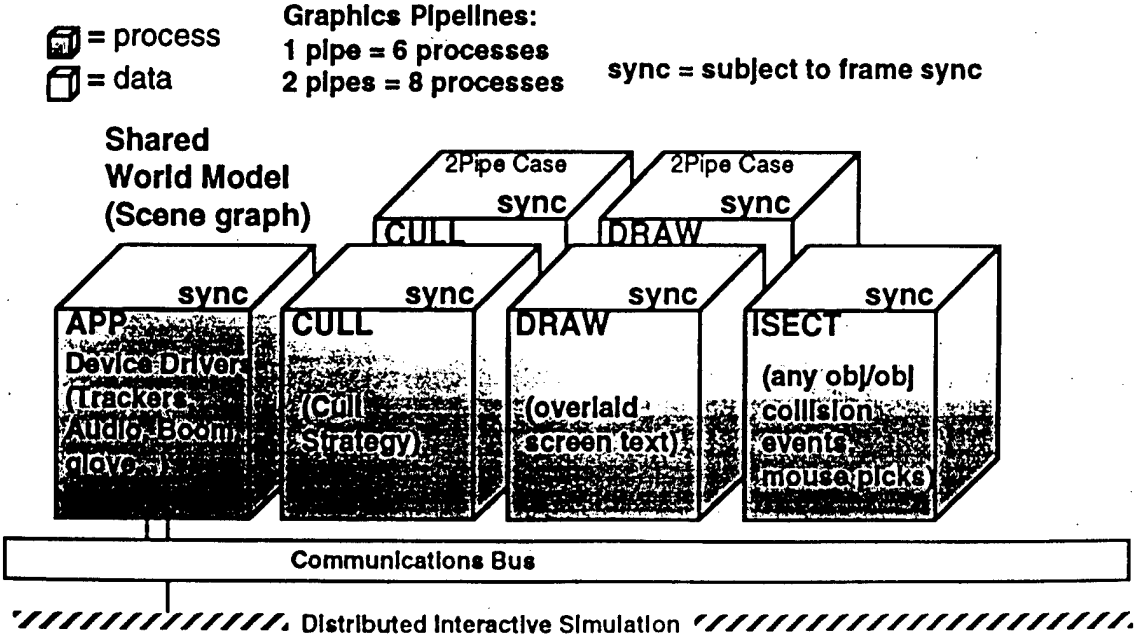
- handle moving participants or controlling views
- handles environment settings such as lighting, background, etc.



When modifying or analyzing the Performer souce code for Vista, it is important to remember that it can run in multi-process mode for increased speed. This has ramifications for how you modify source code, and which bugs you can encounter if done incorrectly. A modification may work fine in single-process mode, but when Vista is run in multiprocess mode, it crashes.

So, any code modifications have to be aware of shared memory used between the processes, and should use the Performer arena for any new memeory structures created. The mult-process figure below illustrates how the Vista Viewer can split out into several processes.

The main.cc source code file in the Vista directory is where this process split is carried out. Before the call to pfConfig() there is one process. After the call to pfConfig() the processes are forked and any ponters no longer work across processes unless they are pointers to shared memory.



Symbols, Abbreviations, and Acronyms

DARPA	Defense Advanced Research Projects Agency
BTL	Behavioral Technologies Laboratories, located in Redondo Beach, CA, a performing organization in the Lockheed Martin VET effort, affiliated with the University of Southern California.
DIS	Distributed Interactive Simulation, a real-time distributed message protocol used in training and operational simulations developed by DARPA and now an International Standards Organization standard.
HTML	HyperText Manipulation Language, a formatting language for pages on the World Wide Web derived from an earlier, more comprehensive document formatting language known as Standard Generalized Manuscript Language (SMGL).
HTTP	HyperText Transfer Protocol, the Internet protocol developed for transport of HTML documents, and now applied to a wide variety of Internet formats, such as VRML
ICAI	Intelligent Computer Aided Instruction, a method of instruction whereby an intelligent model of a student's understanding is used to guide a student during instruction using a computer.
I/ITSEC	Interservice/Industry Training Systems & Education Conference
ISI	Information Sciences Institute in Marina Del Rey, CA, a performing organization in the Lockheed Martin VET effort, affiliated with the University of Southern California in Los Angeles, CA.
ITDM	Intelligent Training Development Methodologies, the contract which is the subject of this final report, which is focused on research and development of instructional methodologies for using intelligent tutoring systems as part of training in a virtual environment.
ONR	Office of Naval Research, the funding agency for the VET effort.
RIDES	Rapid Instructional Development for Educational Simulation
SGI	Silicon Graphics Incorporated, a workstation company whose whole culture centers around fast 3D graphics.
SOAR	A platform independent, cognitive architecture based on a production system which seeks to realize those capabilities required of a general intelligent agent.
Tcl/Tk	a windowing interface toolkit assembled around a unix-shell like interpreter originally developed at UC Berkeley.
TScript	Training Script message protocol for virtual environments
TTRD	Technical Training Research Division, the sponsoring agency of the USAF Armstrong Labs for the ITDM effort
URL	Uniform resource locator, a tag that indicates a media format and location on the Internet as part of the World Wide Web.
VE	Virtual Environment, a 3D visual display and accompanying simulation which represent some aspect of an environment. Expanded forms of VE also address other senses such as audio, touch, etc.
VET	Virtual Environments for Training, a Defense Department focused research initiative funded by the Office of Naval Research, and concerned with applying virtual environment technology to training
VR	Virtual Reality *see Virtual Environment*
VRML	Virtual Reality Modeling Language, an analog to HTML used for documents, but focused on 3D objects and scenes for the World Wide Web.

WWW	World-Wide Web, a system incorporating the HTTP message protocol and the HTML document description language that allows global hypertext over the Internet.