NPS-ME-01-008

NAVAL POSTGRADUATE SCHOOL
Monterey California

EXTENSIBLE 3D (X3D) GRAPHICS: SCENE DESIGN FOR AUTONOMOUS UNDERWATER VEHICLE (AUV) MISSION VISUALIZATION

by

Frederic Roussille

September 2001

Approved for public release; distribution is unlimited

Prepared for: Center for Autonomous Underwater Vehicle (AUV) Research
Naval Postgraduate School, Department of the Navy

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2001		**3. REPORT TYPE AND DATES COVERED** Project report
4. TITLE AND SUBTITLE: Extensible 3D (X3D) Graphics: Scene Design for Autonomous Underwater Vehicle (AUV) Mission Visualization			**5. FUNDING NUMBERS**
6. AUTHOR Frederic Roussille			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Center for Autonomous Underwater Vehicle (AUV) Research, Mechanical Engineering Department, Naval Postgraduate School, Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT

The NPS Center for AUV Research is a leader in this field and has been working for 14 years on several AUV prototypes. Its latest AUV is designated the Acoustic Radio Interactive Exploratory Server (ARIES) and is fully operational.

Because it is sometimes difficult to observe and understand AUV behavior during mission operations, an underwater virtual world can comprehensively model all AUV missions and environment. This report contributes towards real and virtual AUV software development. Indeed, thoughts about a virtual world for AUVs are among the next steps in general AUV development.

This research report is a study and an experiment to transform AUV mission data into visible scenes. These scenes will build up a set of 3D mission archives that could be used later. The chosen programming language is the Virtual Reality Modeling Language (VRML). The programming editor tool is called X3D-Edit, based on XD3 graphics technology and recently upgraded with a French version (French tooltips).

This report also provides 3D VRML/X3D models for the AUV and underwater mine models to improve AUV virtual world realism.

16. SUBJECT TERMS VRML, X3D, Virtual world, AUV			**15. NUMBER OF PAGES** 99
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

This report was prepared for Ecole Nationale des Ingenieurs de Tarbes (ENIT), France.

This report was prepared by:

Frederick Roussille
Ecole Nationale de Ingenieurs de Tarbes

Reviewed by:

Professor Anthony J. Healey
Department of Mechanical Engineering

Associate Professor Don Brutzman
Undersea Warfare Academic Group

Released by:

D. W. Netzer
Associate Provost and
Dean of Research

ABSTRACT

The NPS Center for AUV Research is a leader in underwater robotics and has been working for 14 years on several AUV prototypes. Its latest AUV designated the Acoustic Radio Interactive Exploratory Server (ARIES) and is fully operational.

Because it is sometimes difficult to observe and understand AUV behavior during mission operations, an underwater virtual world can comprehensively model all AUV missions and environment. This report contributes towards real and virtual AUV software development. Indeed, thoughts about a virtual world for AUVs are among the next steps in general AUV development.

This research report is a study and an experiment to transform AUV mission data into visible scenes. These scenes will build up a set of 3D mission archives that can provide post-mission visualization. The chosen programming language is the Virtual Reality Modeling Language (VRML) encoded in Extensible 3D (x3d) Graphics format. The programming editor tool is called X3D-Edit, based on XD3 graphics technology and recently upgraded with a French version (French tooltips).

This report also provides 3D VRML/X3D models for the AUV and underwater mine models to improve AUV virtual world realism.

ACKNOWLEDGEMENTS

I have to thank many people for helping me for this final project.

First of all, I would like to thank Dr. Donald Brutzman, Professor Anthony Healey, Dr. David Marco from the Naval Postgraduate School, M. Thierry Vidal from the Ecole Nationale d'Ingenieurs de Tarbes (France) and the ENIT's international relations for giving the opportunity to come to Monterey (California) for my final project working within the Mechanical Engineering Department of the Naval Postgraduate School.

Secondly, I would like to thank so much again Dr. Donald Brutzman for being my supervisor and for the unconditional support he gave me throughout this project. He answered every question I asked with a constant patience and kindness and always stayed listening to my requests.

I thank M. Thierry Vidal a lot for being my supervisor in France and for the help and advices he gave me during this internship period.

THIS PAGE LEFT INTENTIONALLY BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF DIAGRAMS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Autonomous Underwater Vehicles (AUVs) are designed to independently accomplish complex tasks either in deep oceans or shallow water. A meticulous design must be followed during conception of the AUV, since little or no communication with distant human supervisors is possible. Thus, the underwater domain imposes many limitations and restrictions about hardware and software components selection, as well as hardware and software architecture.

The Center for AUV Research at the Naval Postgraduate School has been working for 14 years on several AUV prototypes, with each improvement showing further success. The latest NPD AUV is called *Acoustic Radio Interactive Exploratory Server* (ARIES) and is fully operational. Currently ARIES operates for short missions in Monterey Bay.

During operations, data sets, gathered from the ARIES include track positions, bathymetry (for each sample point), sonar and video data, contact coordinates, image, etc. All of this data helps to reconstruct what happened during a mission. Nevertheless those information streams are merely data and it is very difficult to observe AUV operations. That is why an underwater virtual world is needed to comprehensively model all AUV missions and all characteristics of the real world where it moves around.

B. MOTIVATION

A virtual world using 3D graphics for the ARIES, provides an excellent design alternative to observe and understand its operations. Because of its high level of realism, a virtual world has the potential to completely change how people observe and analyze post-mission data.

The Virtual Reality Modeling Language (VRML), specially created to design virtual worlds, is a good choice for such tasks. Not only suited to 3D virtual worlds, VRML is also a good way to share information and make these experiments available via the World Wide Web. Extensible 3D (X3D) improvements to VRML provide further benefits.

The main purpose of this project is to demonstrate how transform data into visible information such as the AUV path, AUV models, submerged contact models, etc. Additional functionality is to provide user interactivity during playback: missions displayed “in real time” or not, choosing display parameters, etc. Together These scenes can build up a set of 3D mission archives for long-term use.

C. ORGANIZATION OF THE REPORT

This report is organized into seven chapters:

- Chapter I is the present introduction.
- Chapter II is a presentation of related works pertinent to the ARIES AUV (hardware and software components), and AUV Data Server (ADS) software that collects and transforms mission data from the AUV.
- Chapter III is an overview of VRML and an introductory tutorial to VRML syntax and VRML worlds.
- Chapter IV is a presentation of Extensible 3D (X3D) graphics technology and a X3D based tool, X3D-Edit, used to create VRML/X3D worlds.
- Chapter V describes the VRML/X3D scene, which generates AUV paths according to AUV mission data in a 3D virtual world. This chapter also contains 3D models for individual objects such as the ARIES, underwater mines, etc.
- Chapter VI shows VRML/X3D experimental results derived from AUV operation data.
- Chapter VII provides conclusions and recommendations for future work.

Appendices and associated research products are the final part of this report.

II. RELATED WORK

A. INTRODUCTION

Research on Autonomous Underwater Vehicles (AUVs) has been an ongoing project at the Naval Postgraduate School (NPS) in Monterey, California USA since 1987. Several AUVs followed one another, increasing operational capabilities and becoming more robust as they become more sophisticated in terms of hardware and computer software. The latest NPS vehicle is named *Acoustic Radio Interactive Exploratory Server* (ARIES). This vehicle is a student-research test bed for shallow-water minefield-mapping missions, operating in the littoral ocean. The hull has recently become fully operational, and at the present time, only software enhancements are required. Currently the vehicle operates regularly in Monterey Bay.

The following section is a general overview of the NPS AUV. It provides a general description of the hardware and the software architecture of this vehicle. These descriptions of the ARIES AUV are derived from personal observation and the paper "Current Developments in Underwater Vehicle Control and Navigation: The NPS ARIES AUV" [Marco and Healey, 2001].

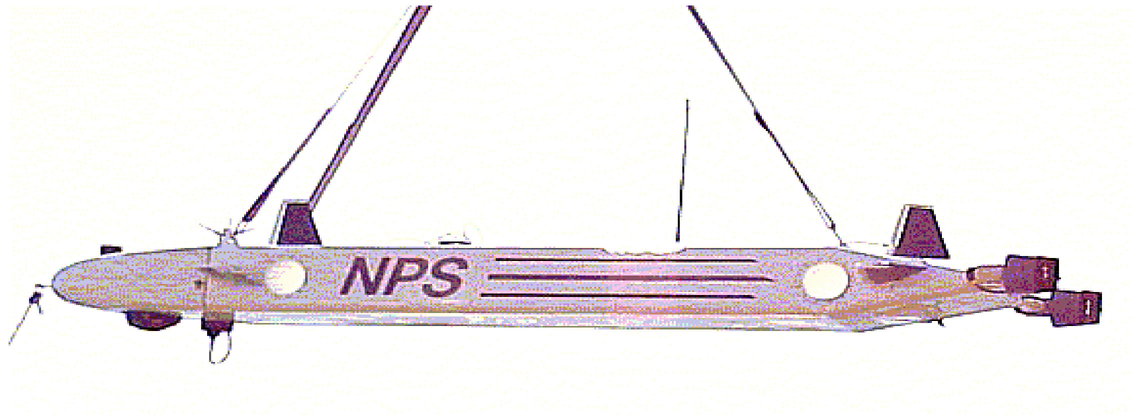


Figure 1. The NPS ARIES AUV "On the Hook," Being Lowered into the Water.

B. VEHICLE PRESENTATION

1. ARIES Hardware

Dimensions and Endurance. The vehicle weighs 225 Kg and measures approximately 3 m long, 0.4 m wide and 0.25 m high. The hull is constructed of 6.35 mm (¼") thick type 6061 aluminum and forms the main pressure vessel that houses all electronics, computers and batteries. A flooded fiberglass nose is used to house the external sensors, key-controlled power "on/off" switches and status indicators. ARIES is capable of a top speed of 3.5 knots and is powered by six 12 volt rechargeable lead-acid batteries. Vehicle endurance is approximately 4 hours at top speed, with 20 hours endurance under hotel load only. The ARIES is primarily designed for shallow water operations and can operate safely down to depths of 30 meters.

Propulsion and Motion Control Systems. Main propulsion is achieved using twin ½ Hp electric drive thrusters located at the stern. During normal submerged flight, heading and depth are controlled using upper bow and stern rudders plus a set of bow planes and stern planes. Since the control fins are ineffective during very slow (or zero) forward-speed maneuvers, vertical and lateral cross-body thrusters are used to control surge, sway, heave, pitch, and yaw, motions [Marco and Healey, 2001].

Navigation Sensors. The sensor suite used for navigation includes a 1200 kHz RD Instruments Navigator Doppler Vedocimeter Log (DVL) that also contains a TCM2 magnetic compass. This instrument measures the vehicle ground speed, altitude, and magnetic heading. Angular rates and accelerations are measured using a Systron Donner 3-axis Motion Pak IMU. While surfaced, Geographic Positioning System (GPS) inputs is provided by a carrier-phase differential GPS (DGPS CP) system available during surfaced operation to correct any navigational errors accumulated during the submerged phases of a mission [Marco and Healey, 2001].

Sonar and Video Sensors. A Tritech ST725 scanning sonar and an ST1000 profiling sonar is used for obstacle avoidance and target acquisition/reacquisition. [Tritech 2001] The sonar heads can scan continuously through 360^{o} of rotation or swept through a predefined angular sector. A fixed-focus wide-angle video camera is located in the nose and connected to a DVC recorder. The computer is interfaced to the recorder which controls on/off and start/stop record functions. While recording images, data for date, time, vehicle position, depth and altitude is superimposed on the video image.

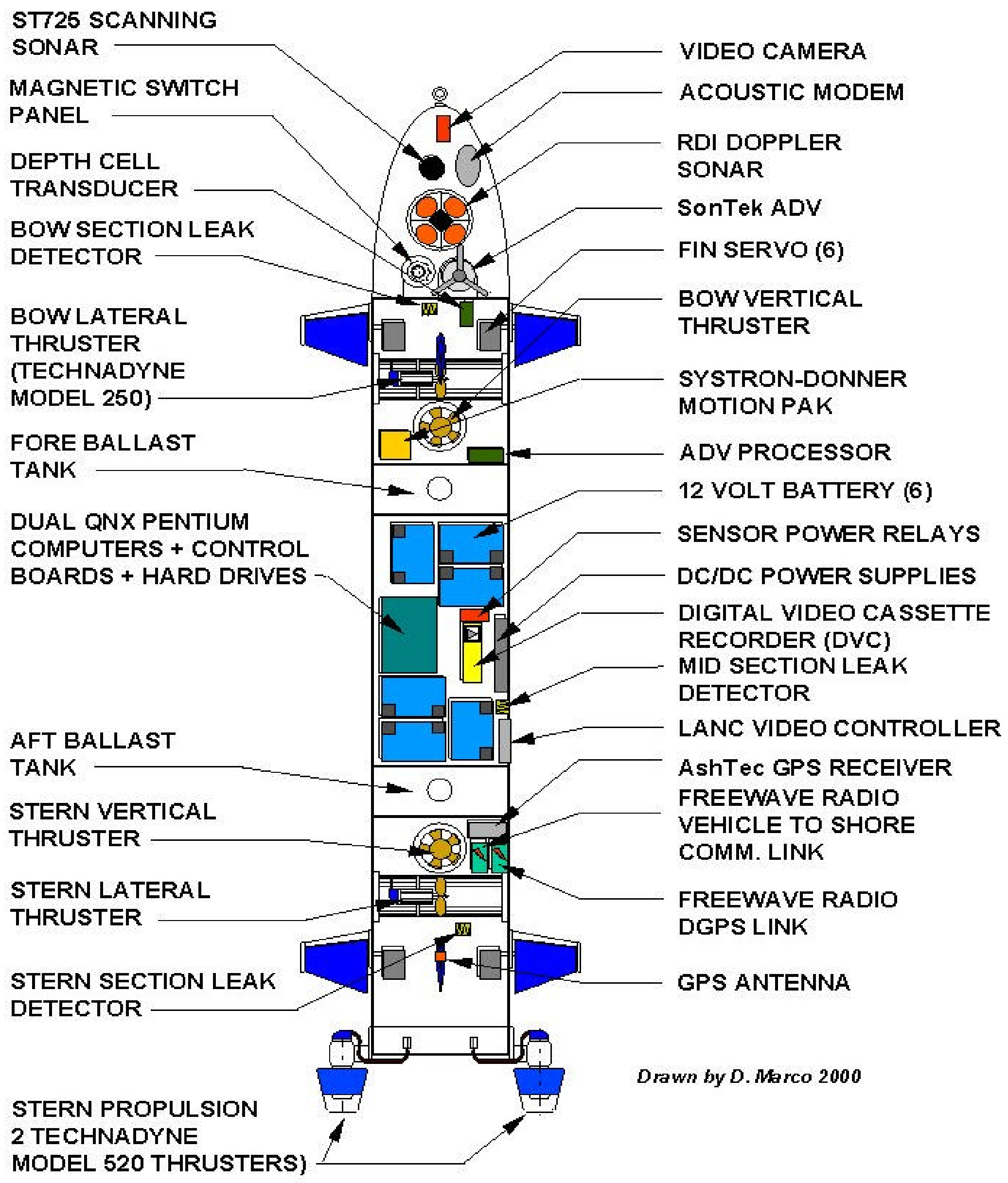


Figure 2. Hardware Components of the NPS ARIES.

Vehicle/Operator Communications. Radio modems are used for high-bandwidth command, control, and system monitoring while the vehicle is deployed and surfaced. While submerged, an acoustic modem is used for low-bandwidth communications. In the laboratory environment, a high-speed thin-wire Ethernet connection is used for software development and mission data upload/download [Marco and Healey 2001].

2. Computer Hardware Architecture

The dual-computer system unit measures approximately 28 x 20 x 20 cm. It consists of two Ampro Little Board 166 MHz Pentium computers with 64 MB RAM, four serial ports, a network adapter, and a 2.5 GB hard drive each. Two DC/DC voltage converters for powering both computer systems and peripherals are integrated into the computer package. The entire computer system draws a nominal 48 Watts [Marco and Healey 2001].

Both systems use TCP/IP sockets over thin-wire Ethernet for inter-processor communications as well as connections to an external LAN. The sensor data-collection computer is designated QNXT. The second is named QNXE and executes the various auto-pilots for servo-level control.

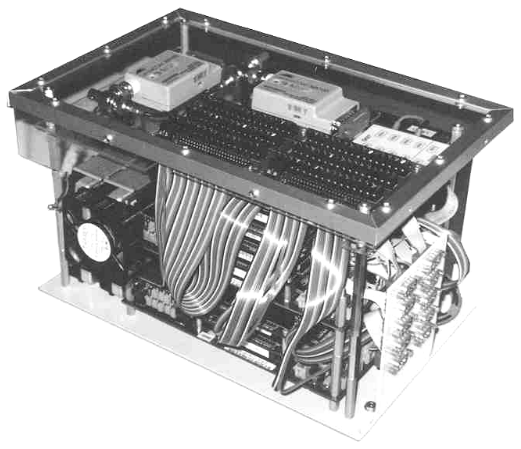


Figure 3. Dual Computer System Unit.

3. Computer Software Architecture.

The *ARIES* AUV has used a tri-level software architecture called the Rational Behavior Model (RBM). RBM divides responsibilities into areas of open-ended strategic planning, soft-real-time tactical analysis, and hard-real-time execution-level control. The RBM architecture has been created as a model of a manned submarine operational structure. The correspondence between the three levels and a submarine crew is shown in the figure below [Lalaque 1999].

Figure 4 represents the tri-level architecture hierarchy with level emphasis and submarine equivalent listed. A functional summary of each level follows.

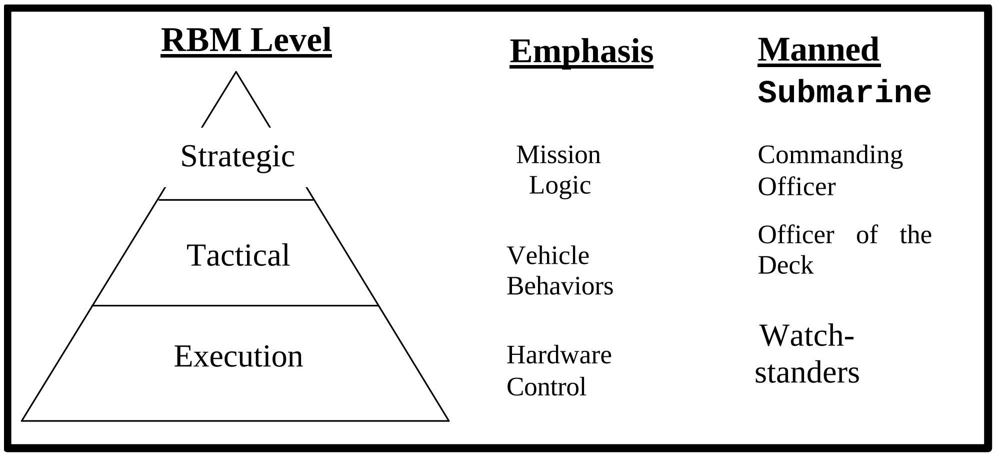


Figure 4. Relational Behavior Model [Holden 1995].

The **Execution Level** assures the interface between hardware and software. Its tasks are to maintain the physical and operational stability of the vehicle, to control the individual devices, and to provide data to the tactical level. These tasks are currently performed by on-board host QNXS [Lalaque 1999].

The **Tactical Level** provides a software level that interfaces with both the Execution level and the Strategic level. Its chores are to give to the Strategic level indications of vehicle state, completed tasks and execution level commands. The Tactical level selects the tasks needed to reach the goal imposed by the Strategic level. It operates in terms of discrete events [Lalaque 1999].

The **Strategic Level** controls the completion of the mission goals. The mission specifications are inside this level [Lalaque 1999].

A diagram outlining the modular, multi-rate, multi-process software architecture is shown in the figure below. The architecture is designed to operate using a single computer processor or two independent, cooperating processors linked through a network interface. Splitting the processing between two computers can significantly improve computational load balancing and software segregation. In the ARIES, each processor assumes different tasks for mission operation [Marco and Healey 2001].

Both computers run the QNX real time operating system (QNX 2001) using synchronous socket sender and receiver network processes for data sharing between the two. Inter-process communication is achieved using semaphore-controlled shared memory structures .

All vehicle sensors are interrogated by separate, independently controlled processes, and there is no restriction on whether concurrent processes operate synchronously or asynchronously. Since various sensors gather data at different rates, each process may be tailored to operate at the acquisition speed of the respective sensor. All processes are written in the C programming language [Marco and Healey 2001].

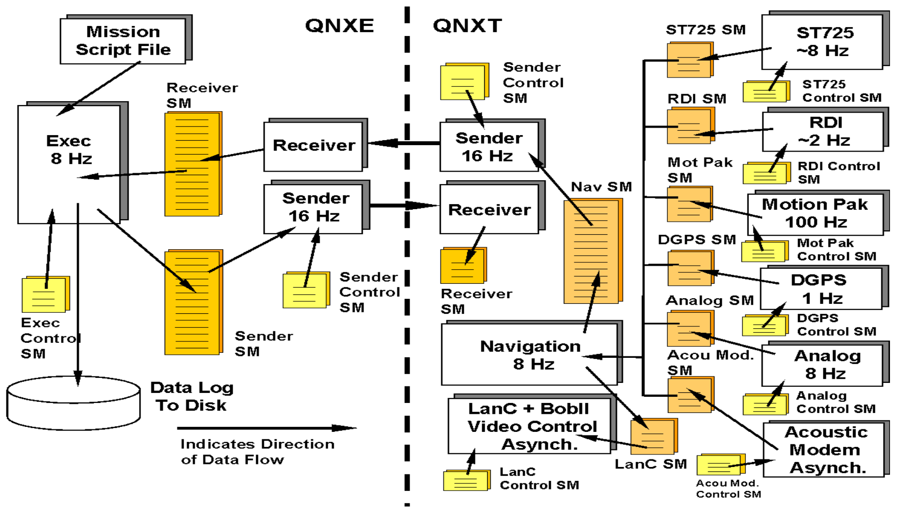


Figure 5. Dual Computer Software Architecture [REF].

To allow synchronous sensor fusion, each process contains a unique shared memory data structure that is updated at the specific rate of each sensor. All sensor data are accessible to a synchronous navigation process through shared memory and is a main feature of the software architecture proposed [Marco and Healey 2001].

C. AUV DATA SERVER (ADS)

ADS is the acronym for *AUV Data Server (ADS)* system. It is a software system developed at NPS and is used to gather and translate AUV data into a format, suitable for input into the *Mine Warfare Environmental Decision Aids Library (MEDAL)* system. This format is used by the U.S. Navy to evaluate asset positions, mine-like contacts, snippet images of those contacts identified as mines, and bathymetry maps. Thus, data gathrered by ADS from the AUV are track positions, bathymetry at each point, sonar and data video processing, image files for contact as well as their locations. Data are converted into *Message Transfer Format* (MTF) message formats and imported into MEDAL [Healey, Wu, Brutzman 2000].

Figure 6 below shows the connectivity for the use of the NPS TDA (the ADS) and its linkage to a stand alone MEDAL station.

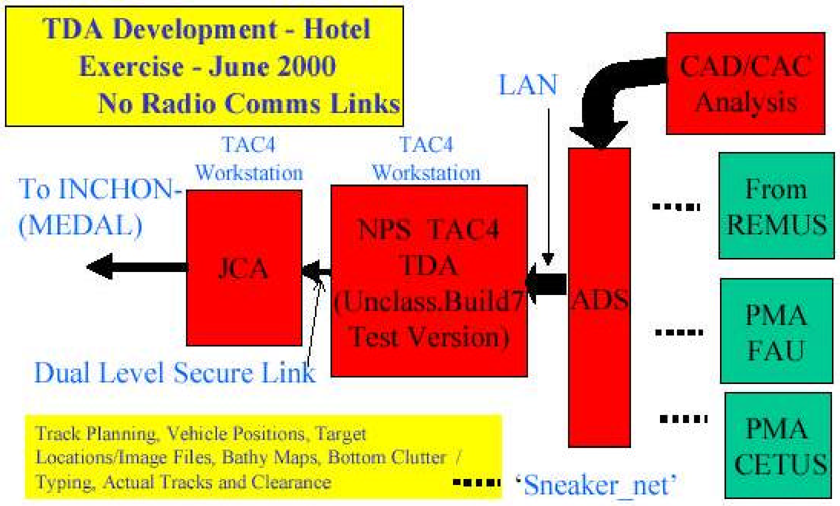

TDA Development - Hotel Exercise - June 2000 No Radio Comms Links
LAN
CAD/CAC Analysis
TAC4 Workstation
TAC4 Workstation
To INCHON-(MEDAL)
JCA
NPS TAC4 TDA (Unclass.Build7 Test Version)
ADS
From REMUS
PMA FAU
PMA CETUS
Dual Level Secure Link
Track Planning, Vehicle Positions, Target Locations/Image Files, Bathy Maps, Bottom Clutter / Typing, Actual Tracks and Clearance
'Sneaker_net'

Figure 6. Block Diagram of the ADS and its Connection to MEDAL.

D. SUMMARY

ARIES AUV hardware and software architectures are described in this chapter. The AUV Data Server (ADS) program used for data gathering is also described.

THIS PAGE INTENTIONALLY LEFT BLANK

III. VRML GRAPHICS

A. INTRODUCTION

This chapter includes two sections. The first is a presentation of the Virtual Reality Modeling Language (VRML). The second section shows and explains how to make a simple VRML scene, constructed with essential VRML nodes.

B. PRESENTATION OF VRML

1. VRML History

The Virtual Reality Modeling Language (VRML) was an idea by Mark Perce and Tony Parisi initially presented at the First International Conference of the World Wide Web in 1994. VRML was intended to be a platform independent language for web-based 3D graphics, and implemented on the Internet. The language needed to be able to place objects in 3D space, as well as include attributes such as shape, color, and size. Since VRML was to be used in the Internet, all platforms needed to be able to support it: UNIX workstations, personal computers, etc. The Silicon Graphics Open Inventor format was the initial basis for the VRML file formats, and after numerous improvements VRML was widely accepted. VRML 1.0 was introduced in 1995. In 1996 VRML 2.0 become the new VRML specification. In 1997, the revised language was certified by the International Organization for Standardization (ISO) as ISO/IEC and was commonly referred to as VRML 97 [Refraction 2001].

2. Presentation

Using VRML, an author can create 3D virtual worlds for display on the web. While VRML 1.0 had static worlds, which is to say that it allowed for no arbitrary behaviors for objects in the VRML world, VRML 97 provides for dynamic behaviors by adding Java and Ecmascript (Javascript) support, as well as sound and animation. The main feature of VRML 97 is that it enables to create dynamic worlds and an interactive environment on the Internet, including the ability to:

- animate objects in the VRML world
- play sounds and movies
- allow users to interact with VRML worlds
- control and enhance worlds with scripts

Since authors are able to create effective 3D virtual worlds, VRML is an appropriate language for moderately complex global scene renderings. Nevertheless VRML is not a Computer Aided Design (CAD) tool. Creating complex shapes with a high level of detail implies using a professional CAD tool like a mechanical engineering program or professional 3D-design software. Nevertheless VRML is a good way for scientists, engineers, hobbyist and application developers to produce composable 3D models for use over the World Wide Web.

3. Browsers and VRML

To present sophisticated multimedia, such as 3D VRML worlds, web browsers (like Microsoft Internet Explorer or Netscape Navigator) need help from compatible applications, called *plug-ins*, that specifically understand content of different filetype formats. They enable users to view non-HTML information within the Web browser window.

Many VRML plug-ins are available as 3D browers, including Silicon Graphics'/Cosmosoftware's Cosmoplayer, Parallel Graphics' Cortona and Blaxxun's Contact browser. VRML remains the preferred language to build non-proprietary virtual worlds and to proesent such work across the Internet.

4. Creating a Simple Object with VRML

Creating a simple scene is a good way to understand the basic principles of VRML syntax. The following example shows different basic nodes and fields for appearance, geometry, sensor and interpolator, ROUTE and viewpoint.

VRML scenes can be created using a simple text editor. More developed VRML editors like X3D-Edit or Parallel Graphics' VrmlPad are highly recommended (especially for the novice).

A VRML 97 file always starts with the line:

```
#VRML V2.0 utf8
```

This is the VRML header, which is required in any VRML file. It must be the first line of the file and it must contain the exact text shown above. The UTF-8 character set (Universal Character Set Transform Format) is a standard way of typing characters in many languages. An example excerpt follow.s

```
Viewpoint {
     description    "First viewpoint"
     position 0 0 20
}
Viewpoint {
     description    "Second viewpoint"
     position 0 0 10
}
```

In a virtual world, the location of a user's viewpoint can be represented by an *avatar*, which is a symbolic virtual-world representation of a real world person. With viewing and navigation represented avatar, the user moves through the virtual world, seeing what the avatar sees and interacting by telling the avatar what to do. The virtual camera representing a user's perspective can see the scene from the position and orientation described by the current *Viewpoint*. The Viewpoint node defines a specific location in the local coordinate system from which the user may view the scene. Authors can create as many viewpoints as desired. Users can navigate through the virtual world by moving from one viewpoint to another, often via the navigation control panel. Viewpoints are important to display object movements or special relationships and it is important to add pertinent Viewpoint nodes when creating complex 3D shapes.

Each Viewpoint collects a variety of related information, described as follows. The *description* field value specifies a text string used to describe the viewpoint. This text string is displayed by the browser control panel. The *position* field specifies a 3D coordinate for the viewpoint location in the current coordinate system. The *Orientation* field describes direction.

The *Shape* node contains the appearance and geometry characteristics of a renderable shape. A typical shape/appearance/geometry example follows.

```
Shape {
  appearance Appearance {
    material DEF SphereColor Material {
      diffuseColor 0 1 0 #Green
    }
       }

  geometry Sphere {
    radius 2 #Meters
```

```
        }
}
```

The *Appearance* node specifies appearance atributes, including the *Material* node. This node includes material attributes as *diffuseColor*, which defines a Red Green Blue (RGB) color for the material. “0 1 0” means that the shape color is full-intensity green with no red or blue color components.

The *Sphere* node is one of the primitive geometry nodes provided by VRML. This node creates a sphere-shaped geometry. In the above example, radius value is 2 meters.

The DEF keyword is used to define a label for a node. For example:

```
DEF ClickOnIt TouchSensor {}
```

This *TouchSensor* node creates a sensor to detect viewer actions and convert them to outputs suitable for triggering actions. The events produced by this particular node (with a DEF name defined as ClickOnIt) are connected to another node via a ROUTE.

The ROUTE written above sends an event from the TouchSensor node (called ClickOnIt by the DEF syntax) to the TimeSensor node (called Clock). IsOver means the value “TRUE” is sent to the TimeSensor when the cursor is over the sphere. The value “TRUE” makes the TimeSensor turn on by sending the value “TRUE” to the Clock node’s field named set_enabled.

```
ROUTE ClickOnIt.isOver TO Clock.set_enabled
```

The *ColorInterpolator* node describes a list of key colors available for use in an animation. The value of the *key* field specifies a list of keys (ranging between 0 and 1) that are used to define relative times matching the functional outputs defined by the *keyValue* field. By retrieving the corresponding pair of key colors to an input key value, the interpolator computes an intermediate interpolating color between the key colors. In this example, corresponding colors to the input keys “0, 0.5, 1” are green, blue, green (“0 1 0, 0 0 1, 0 1 0”).

```
DEF ColorPath ColorInterpolator {
  key [ 0, 0.5, 1 ]
  keyValue [ 0 1 0, 0 0 1, 0 1 0 ] #Green, Blue, Green
}
```

The *TimeSensor* node creates a clock that generates time events to control animations. The *cycleInterval* field specifies the time length in seconds that the TimeSensor takes to vary its fractional time output from fractional time 0 to 1. The *enabled* field specifies whether the TimeSensor is turned on or off. The *loop* field specifies whether the TimeSensor loops (i.e. repeats) or not. The TimeSensor node allows time intervals of arbitrary length, modifying the default time intervals of Interpolators nodes, which are unit length.

```
DEF Clock TimeSensor {
  cycleInterval 3
  enabled FALSE
  loop TRUE
}
```

Behaviors are defined as changing a value in a scene graph. Animation is accomplished by careful design of behaviors, thereby changing parameters of interest. Behaviors are accomplished by event passing: a source value is routed to change another value somewhere in the scene graph. Thus the ROUTEs used in this example animate the sphere. ROUTEs make a one-way circuit to send and receive events between nodes. Each ROUTE remains dormant until an event is sent. A further ROUTE example follows.

```
ROUTE Clock.fraction_changed TO ColorPath.set_fraction
```

Once the TimeSensor is enabled, a time fraction between 0 and 1 is sent from the TimeSensor to the ColorInterpolator node (called ColorPath). The value is put in the *fraction* field and compared to key values. An interpolated color value is their output and sent along the ROUTE. TimeFraction corresponds to the end of the cycle time, in this case 3 seconds.

```
ROUTE ColorPath.value_changed TO SphereColor.set_diffuseColor
```

The interpolated color value is sent to the Material node (called SphereColor). Sphere color is changed, getting this new color value. Finally, it results that when the cursor is over the sphere shape, its color fades from green to blue, and after that, from blue to green and so on until the cursor is no longer over the shape. See Figures 7 and 8.

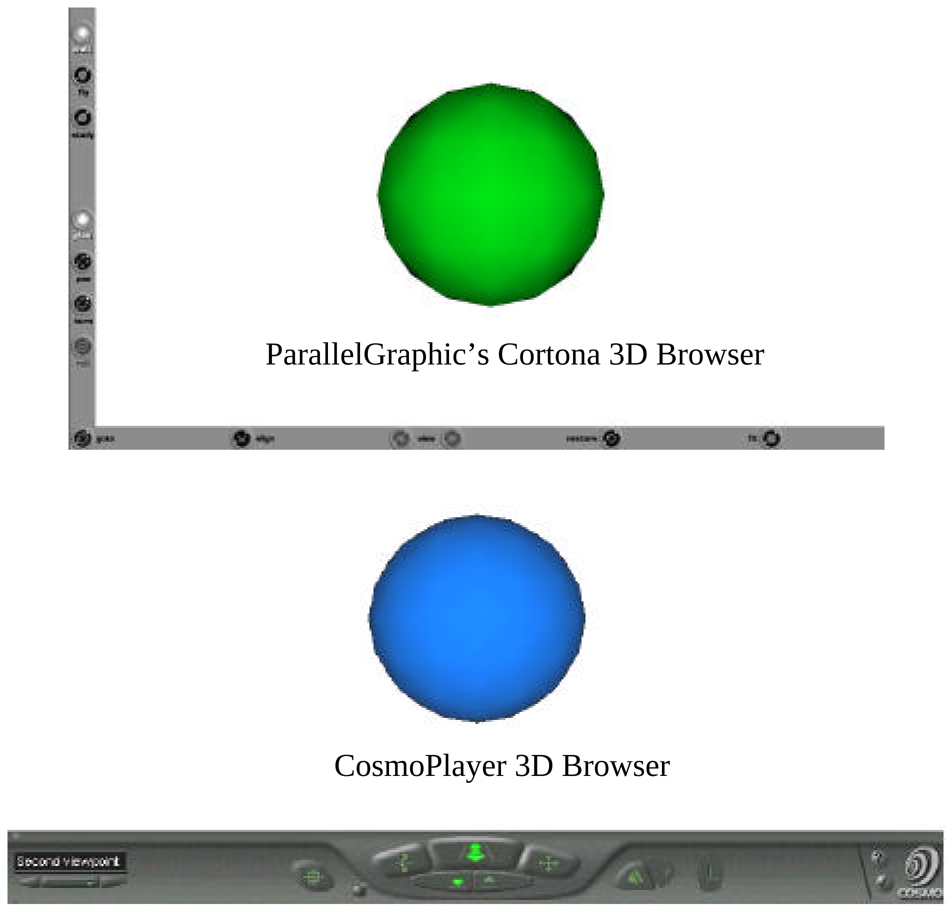


Figure 7. Sphere with Changing Colors Animated Using VRML, Shown in Two Different Browsers. The Color Animates when the Mouse is Over the Object.

C. SUMMARY

The Virtual Reality Modeling Language is presented in this chapter. A simple 3D example-scene shows the capabilities of this language.

IV. EXTENSIBLE 3D (X3D) GRAPHICS

A. INTRODUCTION

This chapter introduces the E-3D (X3D) graphics technology. It includes a presentation of Ext-M-L (XML), the markup language used by X3D graphics tools as well as a presentation of X3D-Edit, an X3D graphics file editor. This section explains briefly how X3D-Edit was made, its main features, and how internationalization support was created.

B. EXTENSIBLE MARKUP LANGUAGE (XML)

Development of the Extensible Markup Language (XML) started in 1996, but in fact the technology isn't completely new. Before XML there was the Standard Generalized Markup Language (SGML), developed in the early '80s. SGML has been an ISO standard since 1986 and is widely used for large documentation projects. The Hypertext Markup Language (HTML), whose development started in 1990 is also originally based on SGML. The designers of XML simply took the best parts of SGML, guided by the experience with HTML, and produced something that is no less powerful than SGML, but vastly more regular and simpler to use [Bosak and Bray 2001].

XML is a markup language for documents containing structured information. Structured information contains different types of content (words, pictures, etc.) and some indication of what role this content plays. As a markup language, it is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents.

XML looks a bit like HTML but is not HTML. Like HTML, XML uses *tags* and *attributes*, but XML uses the tags only to delimit pieces of data, and leaves the interpretation of the data completely to the application that reads it. Thanks to tags and attributes, authors can easily debug applications using a simple text editor to fix a broken XML file. XML isn't meant to be authored by most users but often an XML document can be by deciphered anyone.

Why and when to choose XML? XML was created for richly structured documents that can be used over the web. The only other alternatives, HTML and SGML, are not practical for this purpose.

- HTML is linked with a set of page-presentation layout semantics and does not provide arbitrary structure.

- SGML provides arbitrary structure, but is too complex and difficult to implement for a web browser.

Thus XML is a good choice as a basis for X3D. XML is achieving wide acceptance, which in turn makes more tools available for X3D.

C. X3D-EDIT

1. Overview

X3D-Edit is an Extensible 3D (X3D) graphics file editor that uses the X3D Document Type Definition (DTD) in combination with Sun's Java, IBM's Xeena XML editor building application, and an editor profile configuration file. X3D-Edit enables simple error-free editing, authoring and validation of X3D or VRML scene-graph files. The author of this useful XML editor is Don Brutzman from the Naval Postgraduate School (NPS) [Brutzman 2001].

X3D-Edit is constructed using Xeena, IBM’s tool-building application, and uses Xeena interface [Brutzman 2001]. Xeena is a visual XML editor and a generic Java application for editing valid XML documents derived from any valid DTD. The editor takes as input a given DTD and automatically builds a palette containing the elements defined in the DTD. Users can thus create/edit/expand any document derived from that DTD, by using a visual tree-directed paradigm. Xeena features include:

- Intuitive viewing and editing of X3D documents in a tree control view.
- Editing of multiple X3D documents.
- XML source viewer.
- Direct translation from X3D to VRML 97 syntax using X3D VRML 97.xsl.
- Direct translation from X3D to documentation-quality color-coded HTML.
- Restrictions about adding and editing of features according to the DTD, and validity checking of produced documents.
- Easy customization of display.
- Element-position and attribute-value checking.

Therefore, all those features are automatically included in X3D-Edit. Since X3D-Edit is based on Xeena, users also need to install a Java Development kit (JDK) or Java Runtime Environment (JRE), as Xeena is built on top of Java technology.

2. X3D-Edit Interface

X3D-Edit has a user-friendly interface which is intuitive to use. An action toolbar allows editing/saving/validating XML files. A toolbar palette exposes various node profiles required to build a VRML scene. Major palette sidebar choices include:

- Allowed nodes: context-sensitive display of valid X3D, nodes, fields are available in order to build a valid VRML scene. Nodes appear inside the side bar.
- DIS Java-VRML nodes: the IEE Distributed Interactive Simulation (DIS) protocol is a behavior protocol tuned for physics-based interactions. Java is the programming language used to inplement the DIS protocol, to perform calculations, to communicate with the network as well as the VRML scene. VRML 3D graphics are used to model and render both local and remote entities in shared virtual world.
- Geo VRML: tool created to built complex models of geographic grounds and relief.
- H-Anim: Humanoid Animation Nodes.

Every time an object (node, field, comment, etc) is selected and inserted by the author, it is inserted as directed using a visual tree-directed paradigm into the active document inside the work area. A corresponding attribute array appears in the edit area for the selected node. This is the place where field values are inserted. A message area points out whether there are syntax errors when validating the constructed scene.

It is very easy to build a scene with X3D-Edit because it is possible to copy/paste/move a node or a group of nodes inside the view tree. When you insert a node, only children nodes and fields are available in the sidebar palette so as to avoid fatal syntax errors. Working with a tree paradigm even allows users who do not know the VRML syntax to build complex scenes.

Once the X3D file is created, it can be converted into a VRML file VRML-only browsers. X3D-Edit can make this conversion and launch the VRML browser. It can also convert XML files into pretty-printed HTML files that are easily readable and can be put on the Internet as scene documentation.

X3D-Edit also includes tooltips that helps you to remind the fundamental bases of VRML syntax as well as node/field definition, type, etc.

One ongoing objective for X3D-Edit is to further internationalize context-sensitive node and field tooltips by translating them in many languages (in the profile configuration file).

Currently, English, Spanish and French language tooltips are available and other languages are planned.

To make tooltips in another language different than English, several steps have to be followed. Firstly, a duplication of the file called x3d-compact.profile is necessary (renamed "x3d-compact.profile*LanguageName*"). Within the file, for each "attribute tooltip" tag, (<>) a tooltip sentence is written. This sentence needs to be replaced by a new one with the desired language. Secondly, once the new x3d-compact-profile file created a corresponding. BAT file launching X3D-Edit with the new language version. The code of this BAT file is presented in Figure 8 below:

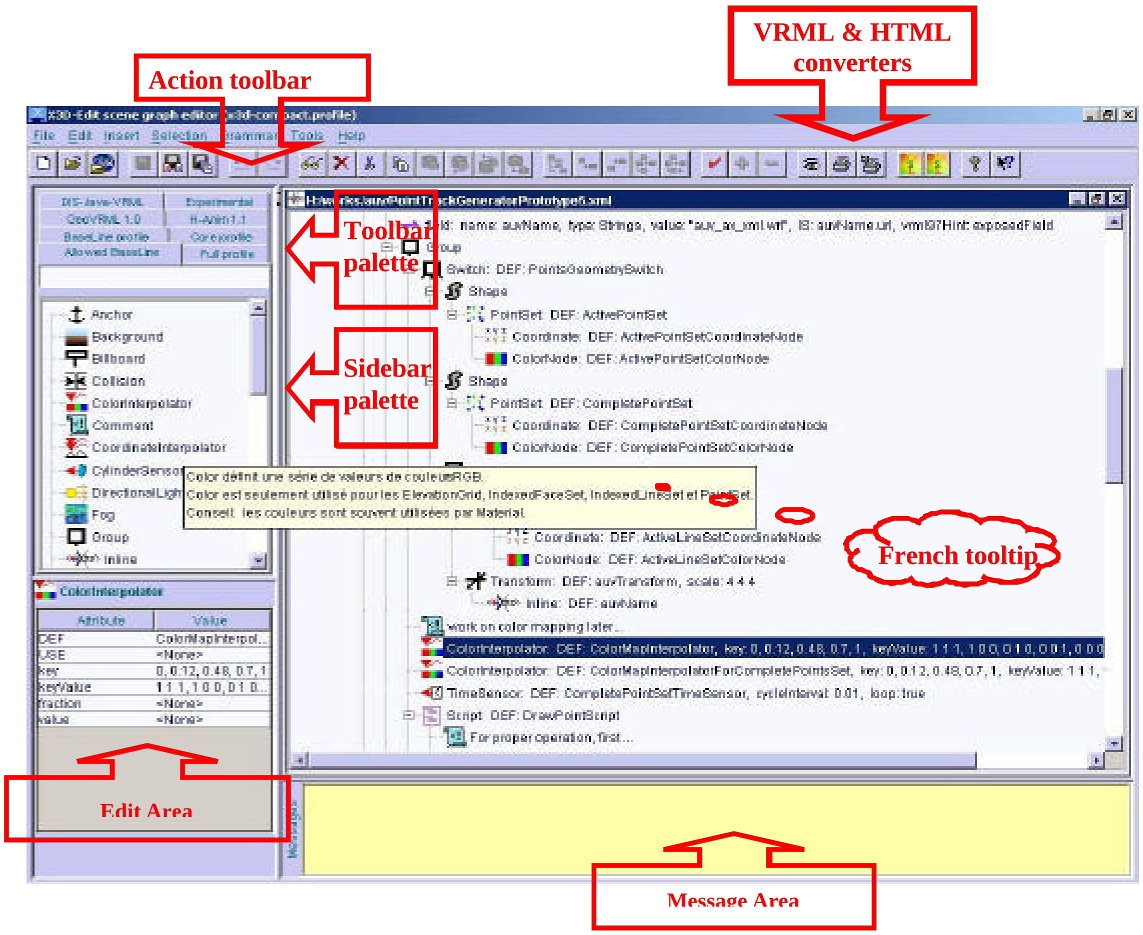
```
@ECHO OFF > NUL

REM   Batch file:  X3D-Edit-LanguageName.bat
REM
http://www.web3D.org/TaskGroups/x3d/translation/X3D-Edit-
LanguageName.bat
REM   Author:      Author name
REM   Revised:     revision date
REM     Description: Launch X3D-Edit profile for x3d-
compact.profileLanguageName

SET X3dLanguagePreference=LanguageName
x3d-edit %1 %2 %3 %4
```

Figure 8. Generic X3D Edit .BAT Template for Different Tooltip Languages.

These two files are put into the X3D-Edit directory. The new BAT file must be run to launch X3D-Edit with the right language version. Figure 9 gives an illustration of X3D-Edit use with French tooktips.



H:\works\auvPointTrackGeneratorPrototype5.xml

field: name: auvName, type: Strings, value: "auv_ax_xml.wrl", IS: auvName.url, vrml97Hint: exposedField
Group
Switch: DEF: PointsGeometrySwitch
Shape
PointSet DEF: ActivePointSet
Coordinate: DEF: ActivePointSetCoordinateNode
ColorNode: DEF: ActivePointSetColorNode
Shape
PointSet DEF: CompletePointSet
Coordinate: DEF: CompletePointSetCoordinateNode
ColorNode: DEF: CompletePointSetColorNode
Coordinate: DEF: ActiveLineSetCoordinateNode
ColorNode: DEF: ActiveLineSetColorNode
Transform: DEF: auvTransform, scale: 4 4 4
Inline: DEF: auvName
work on color mapping later...
ColorInterpolator: DEF: ColorMapInterpolator, key: 0, 0.12, 0.48, 0.7, 1, keyValue: 1 1 1, 1 0 0, 0 1 0, 0 0 1, 0 0 0
ColorInterpolator: DEF: ColorMapInterpolatorForCompletePointsSet, key: 0, 0.12, 0.48, 0.7, 1, keyValue: 1 1 1,
TimeSensor: DEF: CompletePointSetTimeSensor, cycleInterval: 0.01, loop: true
Script: DEF: DrawPointScript
For proper operation, first...

Color définit une série de valeurs de couleurs RGB.
Color est seulement utilisé pour les ElevationGrid, IndexedFaceSet, IndexedLineSet et PointSet.
Conseil: les couleurs sont souvent utilisées par Material.

ColorInterpolator

Attribute	Value
DEF	ColorMapInterpol...
USE	<None>
key	0, 0.12, 0.48, 0.7, 1
keyValue	1 1 1, 1 0 0, 0 1 0...
fraction	<None>
value	<None>

Figure 9. X3D-Edit Interface with Multi-Language Tooltips.

D. SUMMARY

XD3 graphics technology is summarized in this chapter, as well as XML, JAVA and the Xeena software used by X3D-Edit. The X3D authoring tool X3D-Edit is described along with an explanation of how to personalize X3D-Edit tooltips for languages other than English.

THIS PAGE INTENTIONALLY LEFT BLANK

V. MODELING PHYSICAL OBJECTS IN A VIRTUAL OCEAN

A. INTRODUCTION

This chapter describes a virtual environment for the ARIES Autonomous Underwater Vehicle (AUV) of which the main objective is to design a virtual world using the Virtual Modeling Language (VRML). The first section explains the construction of a 3D object modeling the ARIES. The second section presents how the ARIES waypoint tracks and bathymetry can be integrated in the VRML world. Finally, the third section shows authoring of underwater mine contacts, illustrated by a 3D underwater mine example.

B. THE ARIES PROTOTYPE

One of the first VRML models of the Phoenix AUV was built by Don Brutzman and thesis student Martin Whitfield. The ARIES prototype, recently created, is quite similar to the Phoenix model.

The ARIES prototype for this report was designed with X3D-Edit, using VRML/X3D technology. The following paragraphs explain the structure of the 3D AUV object. The body of the AUV is formed by assembling multiple components:

- hull
- propellers
- 8 fins
- Differential Global Positioning System (DGPS)
- NPS logo

1. AUV Hull

The hull is the hardest part of the conception of the AUV model. Because of its complex shape, it has been designed by using an *IndexedFaceSet* node. The *IndexedFaceSet* node is declared inside a S*hape* node. Each 3D point coordinate, constituting the actual shape, is written inside the *Coordinate* node. In total, 38 points are necessary to define the hull. The c*oordIndex* field specifies a connectivity list of coordinate indexes, relative to the points, describing the perimeter of the faces and thus creating the faces. A simple example follows.

```
Shape {
  appearance Appearance {
    material Material {
      diffuseColor 0.9 0.9 0.9
    }
  }
  geometry IndexedFaceSet {
    coordIndex [ Index values here ]
    creaseAngle 3.14
    coord Coordinate {
      point [ Point coordinates here ]
    }
  }
}
```

Note that the *creaseAngle* value gives a smoothly shaded appearance to the hull. When the angle between adjacent polygons exceeds this value, angles formed by adjacent faces appear sharp.

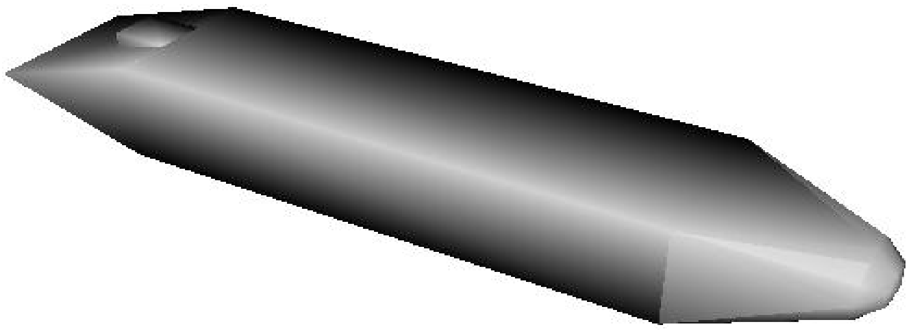


Figure 10. The ARIES Hull with c*reaseAngle* Value = 3.14.

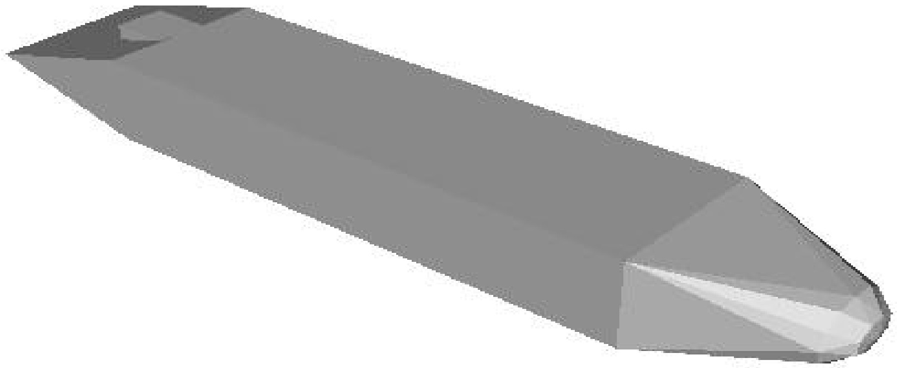


Figure 11. The ARIES Hull with c*reaseAngle* Value = 0.

2. Propellers

As might be assumed, a propeller shape is not so difficult to design. Actually, only one blade is defined and others are replications of the first one. Relative orientations merely change for other blades.

The first blade is built with an *IndexedFaceSet* node. The procedure is strictly the same as the hull. Seven points are defined for this shape. The propeller shaft is a cylinder placed adjacent to an end-cap cone.

The shading cylinder is made with an *Extrusion* node. This requires several steps. First, the *crossSection* field specifies a list of 2D coordinate (on the XZ plane) values that define a section, and is extruded along a spine. Both *scale* and *spine* fields define the path of the extrusion applying a scale factor on the section and that, along each part of the spine. *beginCap* and *endCap* fields indicate whether the beginning and ending faces are drawn or not. In this case, a circular cross-section is extruded along the outer and then inner surfaces of the shroud. Example VRML source follows. A picture of the assembled propeller shroud appears in Figure 12.

```
Shape {
    geometry Extrusion {
      beginCap FALSE
      creaseAngle 2
      crossSection [  1.00  0.00,   0.92 -0.38,
 0.71 -0.71,   0.38 -0.92,
 0.00 -1.00,  -0.38 -0.92,
-0.71 -0.71,  -0.92 -0.38,
-1.00 -0.00,  -0.92  0.38,
-0.71  0.71,  -0.38  0.92,
 0.00  1.00,   0.38  0.92,
 0.71  0.71,   0.92  0.38,
 1.00  0.00 ]
      endCap FALSE
      scale [ 0.08 0.08, 0.07 0.07, 0.06 0.06, 0.07 0.07, 0.08 0.08 ]
      spine [ -0.08 0 0, 0.08 0 0, 0.08 0 0, -0.08 0 0, -0.08 0 0 ]
    }
    appearance Appearance {
      material Material {
        diffuseColor 0 0 1
       }
     }
  }
```

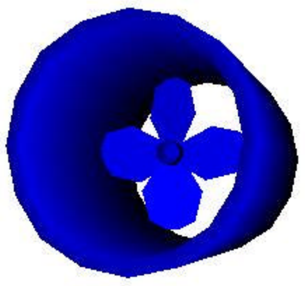
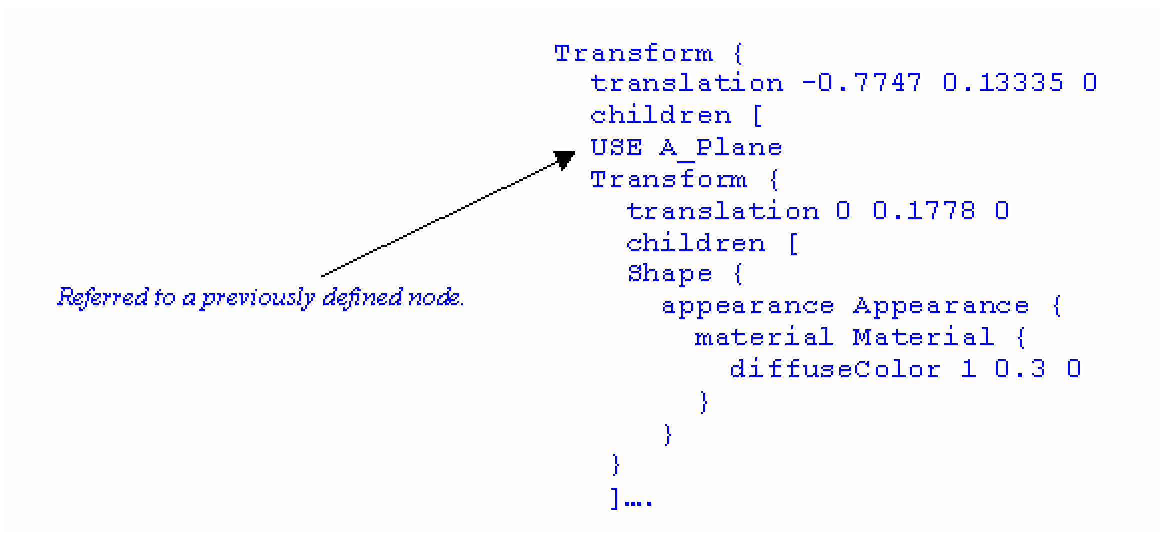


Figure 12. A 3D VRML Propeller inside a Shroud.

3. The Fin and the DGPS

Like the propeller blades and the hull, the fin and DGPS are drawn using the *IndexedFaceSet* nodes. One specified fin shape is defined and replicated seven times. With the syntax *USE* preceding the node name, it is possible to efficiently use a node again and again.



```
Transform {
  translation -0.7747 0.13335 0
  children [
  USE A_Plane
  Transform {
    translation 0 0.1778 0
    children [
    Shape {
      appearance Appearance {
        material Material {
          diffuseColor 1 0.3 0
        }
      }
   }
   ]….
```

4. The NPS Logo

The NPS logo includes several stripes adjacent to text for the acronym “NPS.” Stripes are *IndexedFaceSet* nodes whereas the text uses a *Text* node.

```
Shape {
        appearance Appearance {
          material Material {
             diffuseColor 0 0 0.8
          }
        }
```

```
geometry Text {
 string [ "NPS" ]
 fontStyle FontStyle {
  family [ "SANS" ]
  size 0.15
  style "BOLD"
       }
     }
```

Different fields allow the creation of text geometry: the *string* field specifies lines of text to build, the *FontStyle* node defines the style of the text, etc.

5. The Complete VRML ARIES Model

The figure below shows the VRML model of the ARIES that includes all the different nodes described in previous chapters. Some of them are replicated as USE nodes when needed. A DGPS antenna was added atop the aft upper fin.

This model can be reused in other VRML X3D worlds as desired for example scenes simulating AUV operations.

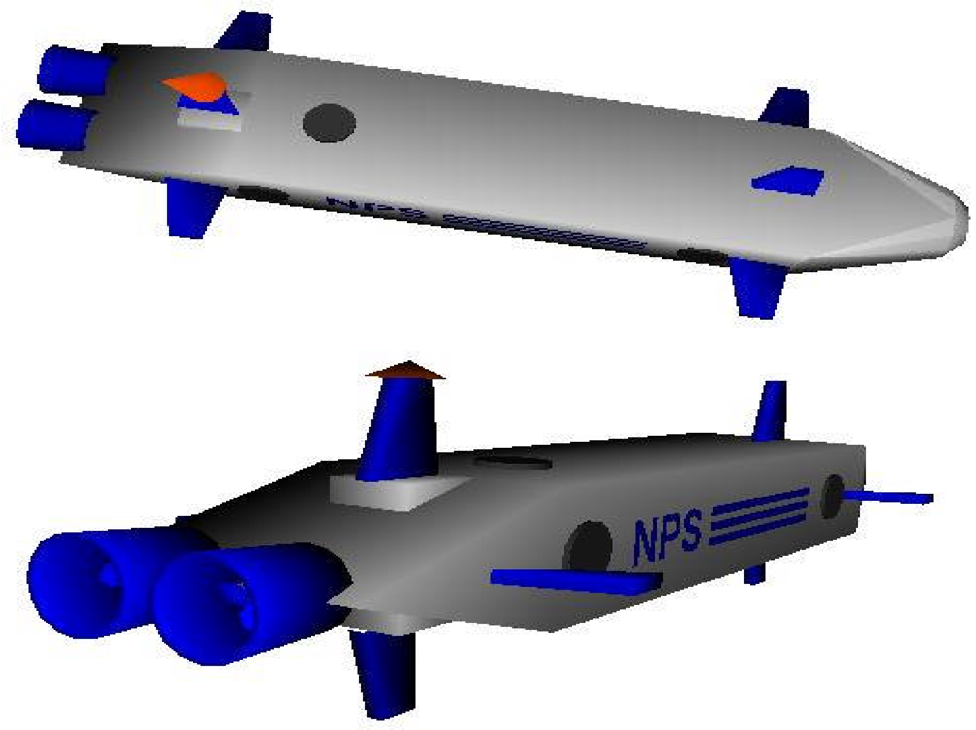


Figure 13. The VRML ARIES Model (Top and Aft Quarter Views).

The complete source code of this model is presented in Appendix A.

6. Another ARIES Prototype

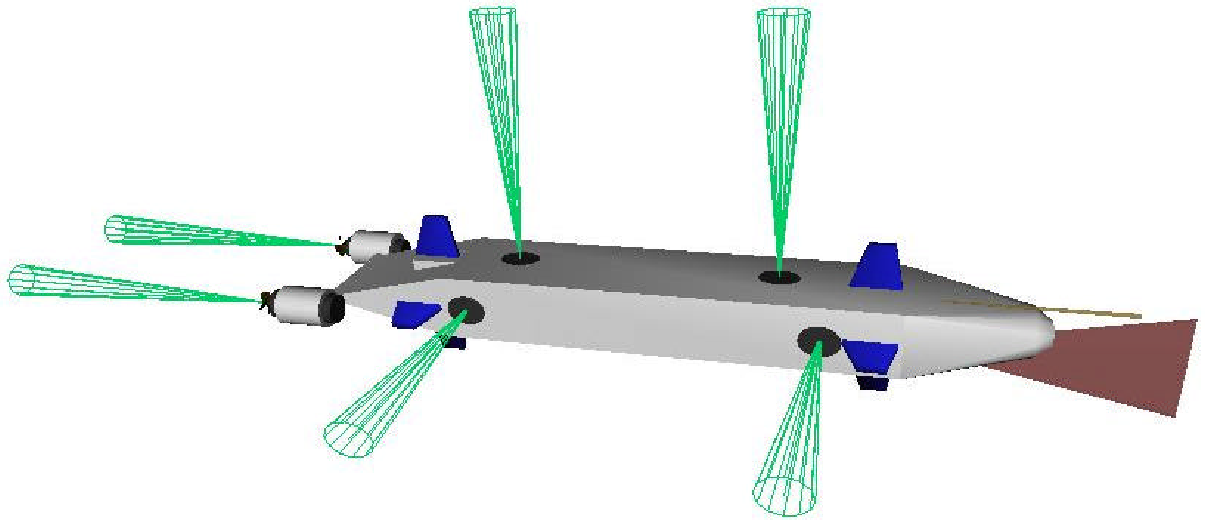


Figure 14. The VRML ARIES Model (Side View).

Jane Wu and Don Brutzman created this improved prototype. The two prototypes are similar because they are based on the same ARIES dimensions. Also included are sonar steering and beam-cone visualizations of thruster flow.

C. THE WAYPOINT TRACK GENERATOR

1. Problem Statement

The purpose of this project is to create a simple scene based on the Virtual Reality Modeling Language (VRML). From inputs that include coordinate data (location + bathymetry) plus time data, the path that followed the AUV during operations needs to be recreated in a 3D virtual world. A simple browser like Netscape Navigator or Internet Explorer might then display this world easily and quickly.

Several alternatives have to be considered. Firstly, the path of the AUV is displayed with a large quantity of points that are coordinate points. Each point is colored, to indicate the depth of the AUV at this point. All the points are displayed at the same time, without caring about the time references in the point list.

Secondly, the path of the AUV is similarly displayed with a quantity of points that are coordinate points, but each point is associated to a time reference (also called “time fractions” or “time stamps”). The bathymetry is still symbolized by colors but points appear sequentially according to their respective time fractions, matching the "real time" of the original data collection.

Thirdly, the path is shown with line segments with a single color. There is no color representation of the bathymetry. Lines are drawn according to their respective time fractions as the second solution.

Coordinate points come from a data file generated by the ADS software (described in Chapter II). ADS further constructs complete VRML X3D scenes for each mission using these 3D models, producing a set of mission archive 3D scenes.

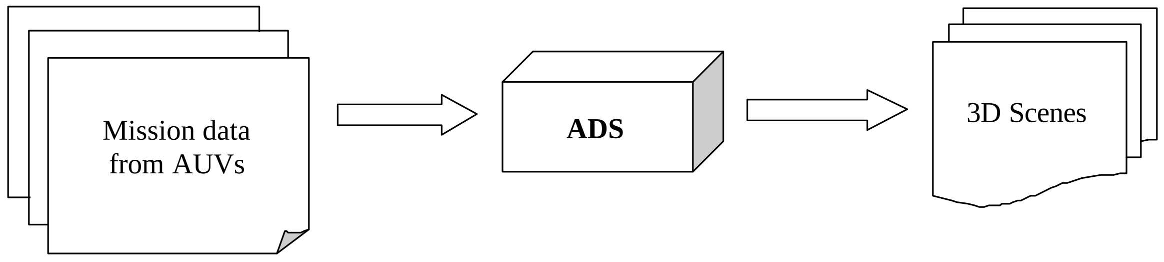


Figure 15. The ADS Software Analyzes AUV Mission Data and Produces 3D Scenes using the 3D Models Presented in this Chapter.

2. Designing the Waypoint Track Generator Prototype

a. Prototype Declaration and Instantiation

A ProtoDeclare declaration defines a new Prototype node. Like any other VRML/X3D node, a ProtoDeclare can contain *fields*, *eventIn*, *eventOut*, *Shapes*, *Groups*, interpolators and more. A Prototype node can be reused by external VRML scenes as often as wished, through instantiation using ProtoInstance nodes. By specifying field values, it is easy for authors to change ProtoInstance properties and thus to configure these new nodes at run time. For these reasons, Prototypes are used to define the customizable waypoint track generator.

A ProtoInstance statement instantiates a prototype node in the scene. It is comprised of two major parts: the node interface and the body that contains other nodes. The node interface includes four types of fields:

- *field* defines variables that have no interaction with the outside (ROUTEs or script code).

- *eventIns* are receiver variables, which wait for events from the outside and take them in to be handled.
- *eventOut* are transmitter variables, which send events from the node to the outside.
- *exposedFields* are essentially a combination of field eventIn and eventOut functionality.

```
field          MFVec3f      pointPositionsArray [ 0 0 0, 10 -4 0, 25 -
6 0, 30 -8 5, 38 -15 5, 45 -18 5, 55 -22 5, 60 -25 15, 60 -27 22, 55 -
30 35, 48 -35 35, 35 -35 35, 25 -45 35, 20 -55 35, 15 -70 35, 3 -70
35, -5 -72 40, -5 -75 50, 0 -80 55, 15 -75 55, 30 -70 55, 35 -60 55,
40 -50 55, 50 -34 55, 65 -23 70 ]
# pointPositionsArray provides point coordinates in meters, referenced
to local coordinate system origin
field           MFTime      pointTimesArray [ 1, 3, 6, 8, 10, 12, 14,
15, 17, 18, 23, 28, 35, 37, 39, 43, 45, 47, 48, 53, 58, 60, 61, 65, 70
]
# pointTimesArray provides point times in seconds for local exercise
clock (each time is clock time in seconds, not in interval durations).
# Both point coordinates and times are initially provided as a full
set of values.
eventOut          SFTime      totalDuration
# totalDuration is derived from the pointTimesArray, and used to set
cycleInterval on a controlling TimeSensor clock outside the
PointTrackGenerator ProtoInstance.
exposedField      SFInt32     displayPointsMode  0
# displayPointsMode settings:  -1 = none, 0 = some points (active
interval, default value), 1= all the points, 2 = some lines (active
interval).
  eventIn      SFTime      durationActivePoints
  # durationActivePoints is in seconds, default initialization value
is totalDuration
  eventIn      SFTime     timeLatestActivePoint
  # timeLatestActivePoint is in seconds, default initialization value
is the final point time
  eventOut     SFTime     getStartTime
  # getStartTime is the time when the TimeSensor must start
  eventOut     SFTime     getStopTime
  # getStopTime is the time when the TimeSensor must stop
  eventIn      SFTime     mappedColorPointCreator
 # mappedColorPointCreator is a function receiving the time fractions
of the TimeSensor
  exposedField MFString   auvName     ["auv_ax_xml.wrl"]
 # auvName refers to an external VRML file name that has 3D AUV models
]
```

b. *Prototype Body: The Switch Node*

The Switch node gives a choice to select one of several different groups of nodes, each contained as distinct children within the Switch node. The VRML browser displays only one shape (or group of shapes) selected. By providing different visualization possibilities as distinct children of a Switch node, the Waypoint Track Generator can render waypoints in different ways.

Using this node is important for the scene to have the choice in the way to display the AUV path (points, lines, linked to time fractions or not). *whichChoice* starts with 0 for the first child. If no childe is desired, *whichChoice* can be set to –1. Note that all children of the Switch node continue to receive and send events regardless of the choice specified by *whichChoice*.

Inside the Switch node body are three nodes that can independently display points or line for the visual reproduction: two *PointSet* nodes and one *IndexedLineSet* node. The PointSet node creates point geometry while the IndexedLineSet node creates polyline geometry. In this scene, the Switch node is followed by an Inline node, which opens a VRML file specified by the URL and renders its contents. On that account, it is easily feasible to include whatever is wanted into the scene (e.g. any types of AUV or even other types of vehicles) which might follow the lines.

c. *Prototype Body: Interpolators and Sensor Nodes*

Two *ColorInterpolator* nodes are required for assigning colors to points. From a set of referenced colors (compared to a set of depth values), these nodes interpolate the input depth value and compute an RGB color, which can be sent to other nodes or variables. This is a concise and efficient way to map arbitrary values (such as depth) to a color value.

The scene use two ColorInterpolator nodes to avoid confusion when the different script functions send values to the interpolators at the same time.

A *TimeSensor* node generates time events to control a script function: actually, this sensor is really useful or else the script function (*completePointSetValue_changed*) cannot start itself. Thus the TimeSensor drives the simulation clock and has a cycle time corresponding to the time duration of a complete traversal.

d. *Prototype Body: Script Nodes*

Script nodes are essential to perform complex actions. They receive input, process computations and avoid output values to Interpolators, Sensor and Shape nodes in the scene. There is only one essential script node in this prototype. It contains many different functions that are the scene core. This script node is named *DrawPointScript*.

A bit like a Proto Instance, a Script node has a field declaration for variable initialization and declaration, and the URL field that encloses EcmaScript (Javascript) source-coded functions. In a Script node, declaration types are field, eventIn and eventOut. Note that exposeField is not allowed in Script nodes, which is a significant inconvenience and will hopefully be changed in future version of the X3D spec.

```
field   MFVec3f  pointPositionsArray IS pointPositionsArray
# This variable is linked to pointPositionsArray from the Proto
Instance field declaration
field   MFTime   pointTimesArray IS pointTimesArray
# This variable is linked to pointTimesArray from the Proto Instance
field declaration
field   MFVec3f  newPointPositionsArray [ ]
# It is a new point coordinate array when the last point coordinate
chosen is different from the latest pointPositionsArray coordinate.
Otherwise, newPointPositionsArray = pointPositionsArray by default
field   MFTime   newPointTimesArray [ ]
# It is a new time fraction array when the last time fraction chosen
is different from the latest newPointTimesArray fraction. Otherwise,
newPointTimesArray = pointTimesArray by default
field   SFInt32  lineIndex 1
# Integer that is incremented to add new values in coordIndex_changed
array
eventIn SFTime   mappedColorPointCreator IS mappedColorPointCreator
# This variable is linked to mappedColorPointCreator from the Proto
Instance field declaration

field   SFInt32  index 0
# Integer that is incremented to add new values in several arrays
(coordinate, time, color arrays)

field   SFInt32  completeIndex 0
# Integer that is incremented to indicate how much coordinate and
colors values have to be added
eventOut   SFBool    ConditionComplete
# Boolean that stops function working during ColorInterpolator
processing
field      SFNode  ActivePointSetCoordinateNode USE
ActivePointSetCoordinateNode
# this variable is linked to ActivePointSetCoordinateNode and allows
to acquire and put values in it
```

```
field      SFNode   ActivePointSetColorNode   USE
ActivePointSetColorNode
# this variable is linked to ActivePointSetColorNode and allows to
acquire and put values in it
FIELD      SFNODE   COMPLETEPOINTSETCOORDINATENODE USE
COMPLETEPOINTSETCOORDINATENODE

# this variable is linked to ActivePointSetColorNode and allows to
acquire and put values in it
field      SFNode   CompletePointSetColorNode USE
CompletePointSetColorNode
# this variable is linked to CompletePointSetCoordinateNode and allows
to acquire and put values in it
field      SFNode    ColorMapInterpolator USE ColorMapInterpolator
# this variable is linked to ColorMapInterpolator and allows to
acquire and put values in it
FIELD      SFNODE    COLORMAPINTERPOLATORFORCOMPLETEPOINTSSET USE
COLORMAPINTERPOLATORFORCOMPLETEPOINTSSET

# this variable is linked to ColorMapInterpolatorForCompletePointsSet
and allows to acquire and put values in it
field      SFNode    ActiveLineSetCoordinateNode USE
ActiveLineSetCoordinateNode
# this variable is linked to ActiveLineSetCoordinateNode and allows to
acquire and put values in it
field      SFNode    ActiveLineSetColorNode USE ActiveLineSetColorNode
# this variable is linked to ActiveLineSetColorNode and allows to
acquire and put values in it
field      SFNode    auvTransform USE auvTransform
# this variable is linked to auvTransform and allows to acquire and
put values in it
eventOut   SFTime    totalDuration IS totalDuration
# This variable is linked to totalDuration from the Proto Instance
field declaration
eventOut   SFTime    getStartTime IS getStartTime
# This variable is linked to getStartTime from the Proto Instance
field declaration
eventOut   SFTime    getStopTime IS getStopTime
# This variable is linked to getStopTime from the Proto Instance field
declaration
eventOut   MFInt32   coordIndex_changed
# This is a coordinate array used for the ActiveLineSetCoordinateNode
node
eventIn   SFTime   durationActivePoints IS durationActivePoints
# This variable is linked to durationActivePoints from the Proto
Instance field declaration
eventIn   SFTime   timeLatestActivePoint IS timeLatestActivePoint
# This variable is linked to timeLatestActivePoint from the Proto
Instance field declaration
eventIn   SFTime   completePointSetValue_changed
# This function computes depth values from coordinate values
eventIn   SFColor  set_completePointSetColorArray
```

```
# Function that puts coordinate and color values in arrays for the
CompletePoinSet Node
```

(1) The function `initialize()`. After the field declaration follows the JavaScript code written inside the URL field (or a containing CDATA block in X3D form). The code consists of four functions. The function ***`initialize()`*** is the first function of the code to be read by the browser and the only one to start without receiving an output value. All initializations and setup choices for the scene rendering are made inside this function.

```
javascript:

function initialize() {
 totalDuration = pointTimesArray[pointTimesArray.length-1];
 var today = new Date();
 getStartTime = Math.round(today.getTime() / 1000);
 getStopTime = getStartTime + totalDuration;
 var m = 1;
```

totalDuration is the AUV operation total duration. Its value is equal to the last time fraction of the *pointTimesArray*. Thus initial value of pointTimesArray should be zero.

getStartTime takes the current time value when the code line is read. As *getTime()* returns a value in milliseconds, *Math.round(today.getTime() / 1000)* is used to convert the value in seconds. The *getStartTime* value served as time reference for the TimeSensor called *DisplayingTimer*, located outside the Proto Instance. This sensor sends time values to the eventIn function *mappedColorPointCreator*.

getStopTime is equal to the *getStartTime* plus the total duration. It is the time when the TimeSensor must stop.

```
//default values for durationActivePoint and timeLatestActivePoint
 durationActivePoint = totalDuration;
 timeLatestActivePoint = pointTimesArray[pointTimesArray.length-1];
```

durationActivePoint and *timeLatestActivePoint* allow to the user to choose what range of values to display on screen. *durationActivePoint* gets the totalDuration value by default (it means all the points are considered). It is possible to reduce the interval of values as wished. timeLatestActivePoint is the latest time fraction (and thus also latest point coordinate) that has to be displayed on the scene. The default value is the last time fraction of pointTimesArray.

```
if(timeLatestActivePoint == durationActivePoint) {
  newPointTimesArray = pointTimesArray;
  newPointPositionsArray = pointPositionsArray;
  print('newPointTimesArray = ' + newPointTimesArray);
 }
```

newPointTimesArray and *newPointPositionsArray* are clones of pointTimesArray and pointPositionsArray when values are chosen by default. Those two new variables will be used to feed the coordinate and color fields of ActivePointnode and ActiveLineSetNode.

```
if(timeLatestActivePoint > durationActivePoint) {
  var firstTime = latestTime = k = 0;
  while((timeLatestActivePoint - durationActivePoint) !=
pointTimesArray[firstTime]) {
   firstTime++;
  }
  while(timeLatestActivePoint != pointTimesArray[latestTime]) {
   latestTime++;
  }
  for(var j = firstTime ; j <= latestTime ; j++) {
   newPointTimesArray[k] = pointTimesArray[j] -
pointTimesArray[firstTime] + 1;
   newPointPositionsArray[k] = pointPositionsArray[j];
   k++;
  }
  print('newPointTimesArray = ' + newPointTimesArray);
 }
```

If timeLatestActivePoint and durationActivePoint get new values, newPointTimesArray and newPointPositionsArray must change and receive new values. This condition searches for the first time fraction (last specified time fraction minus specified duration) and puts selected values in the new newPointTimesArray and newPointPositionsArray arrays.

"newPointTimesArray[k] = pointTimesArray[j] - pointTimesArray[firstTime] + 1".

Means the first point chosen by the user is not displayed according to its corresponding time but starts at the beginning (users don't have to wait until the right time). All new time values are shifted. Note the value "1" is added; it is a trick and otherwise the program is in trouble since it has no prior time to compute properly the first values.

```
if(timeLatestActivePoint < durationActivePoint) {
  print('Fatal error : timeLatestActivePoint < durationActivePoint
!');
 }
 ConditionComplete = false;
}
```

The last condition helps the user if he makes a mistake by reminding him that timeLatestActivePoint < durationActivePoint is a nonsense condition. Error messages appear in the 3D browsers VRML console.

(2) The function completePointSetValue_changed()

```
function completePointSetValue_changed() {
 if(ConditionComplete == false && completeIndex <=
(pointPositionsArray.length-1)) {
  ColorMapInterpolatorForCompletePointsSet.set_fraction = -
pointPositionsArray[completeIndex][1] / 100;

print('ColorMapInterpolatorForCompletePointsSet.set_fraction['+complet
eIndex+'] = ' +
ColorMapInterpolatorForCompletePointsSet.set_fraction);
 ConditionComplete = true;
 }
}
```

This function is written to put a modified value, which symbolizes the depth value of a specified point, in the ColorInterpolator called *ColorMapInterpolatorForCompletePointsSet*. This one uses a list a key values and key colors in its *key* and *keyValue* fields. When it receives a value, it uses linear interpolation to compute intermediate colors. The input fraction value is usually within a range of [0…1], which is why depth values are converted.

The function *completePointSetValue_changed()* cannot be self-executed and cannot work several times without an external input event. Its field is SFTime. A

TimeSensor (CompletePointSetTimeSensor) controls the execution of this function by sending time signals all the time.

All depth values are computed one after the other, waiting until the color interpolator finishes to compute color value, thanks to the Boolean *ConditionComplete* (loop if inactive until ConditionComplete = false).

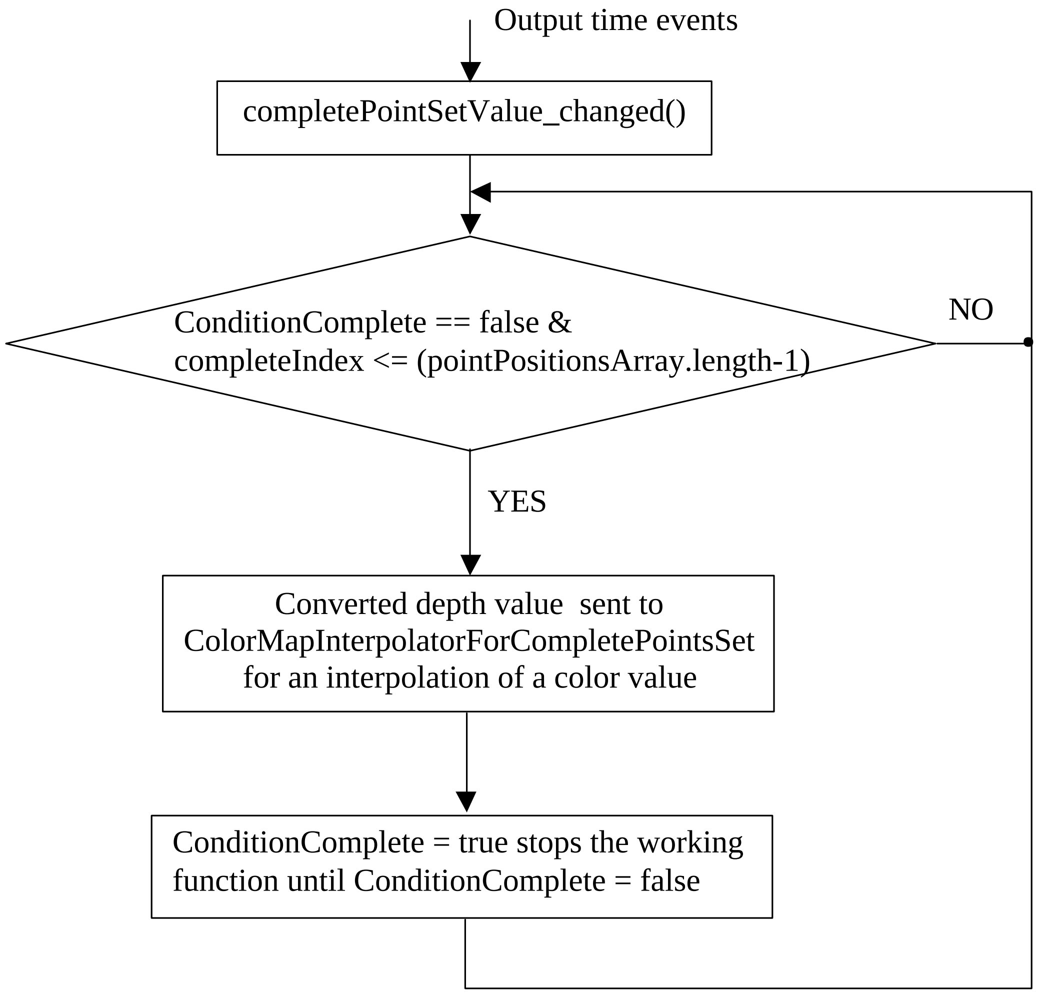

Output time events
completePointSetValue_changed()
ConditionComplete == false &
completeIndex <= (pointPositionsArray.length-1)
NO
YES
Converted depth value sent to ColorMapInterpolatorForCompletePointsSet for an interpolation of a color value
ConditionComplete = true stops the working function until ConditionComplete = false

Diagram 1. Flowchart Function for the Function CompletePointSetValue_changed().

(3) The function set_completePointSetColorArray(). After computing, the color interpolator sends a color value, which travels through a ROUTE to the eventIn set_completePointSetColorArray() function. The color value is stored in the CompletePointSetColorNode color array. Point coordinates are also stored in the

CompletePointSetCoordinateNode point array. Thus, thanks to coordinate and color values, a colored point is displayed on screen.

The ConditionComplete value changes to reactivate the if() condition inside the completePointSetValue_changed().

```
function set_completePointSetColorArray(Value) {
  CompletePointSetColorNode.color[completeIndex] = Value;
  CompletePointSetCoordinateNode.point[completeIndex] =
pointPositionsArray[completeIndex];
  completeIndex++;
  ConditionComplete = false;
}
```

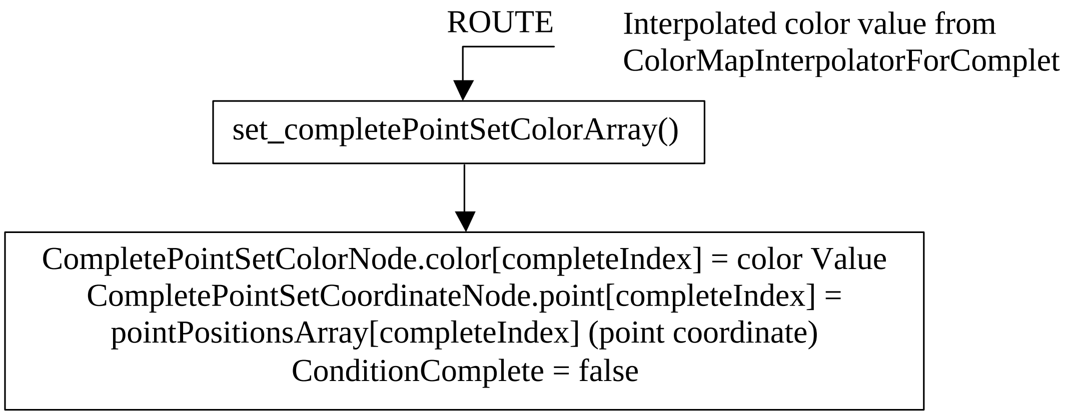

ROUTE
Interpolated color value from ColorMapInterpolatorForComplet
set_completePointSetColorArray()
CompletePointSetColorNode.color[completeIndex] = color Value
CompletePointSetCoordinateNode.point[completeIndex] =
pointPositionsArray[completeIndex] (point coordinate)
ConditionComplete = false

Diagram 2. Diagram of the Function set_completePointSetColorArray().

(4) The function mappedColorPointCreator(). This function provides coordinate and color values for ActivePointSet and ActiveLineSet nodes. This is the biggest function of the DrawPointScript script.

MappedColorPointCreator receives continuously time fractions sent by a TimeSensor (Displaying Timer) thanks to a ROUTE. The received value is compared to the time fractions of the newPointTimesArray array. If one of them is equivalent, it means the point corresponding to the time fraction has to be displayed on screen. Then the condition if() is executed. Note that the input time value is rounded because it cannot match exactly with a time fraction (in newPointTimesArray) if not.

```
function mappedColorPointCreator(fractionValue) {

  ColorMapInterpolator.set_fraction = -
newPointPositionsArray[index][1] / 100;
 /need to initialize ColorMapInterpolator.set_fraction with the first
point color otherwise the          value is shifted
if(Math.floor(fractionValue) == (newPointTimesArray[index] +
getStartTime)) {
 ActivePointSetColorNode.color[index] =
ColorMapInterpolator.value_changed;
 ActivePointSetCoordinateNode.point[index] =
newPointPositionsArray[index];
 auvTransform.translation = newPointPositionsArray[index];
```

A converted depth value is put in the color interpolator node for color interpolation. When the output color is interpolated, it is stored in the color field of the *ActivePointSetColorNode* node. At the same time point coordinate is put in the point fields of the *ActivePointSetCoordinateNode* node and of the *ActiveLineSetCoordinateNode* node. This coordinate value also feeds the position field of the AUV transform node. In this way, the AUV will move each time a new point is added. Nevertheless, the motion of the AUV will appeaer jerky because there is no interpolation for it (this choice could be conceivable in future versions).

```
   if(index <= 1) {
        ActiveLineSetCoordinateNode.point[index] =
newPointPositionsArray[index];
              coordIndex_changed[index] = index;
        ActiveLineSetColorNode.color[index][0] = 1;
        ActiveLineSetColorNode.color[index][1] = 1;
        ActiveLineSetColorNode.color[index][2] = 1;
        auvTransform.translation = newPointPositionsArray[index];
   if(index == 1) {
                ActiveLineSetCoordinateNode.point[index] =
newPointPositionsArray[index];
            coordIndex_changed[index] = index;
            coordIndex_changed[index+1] = -1;
            ActiveLineSetColorNode.color[index-1][0] = 1;
            ActiveLineSetColorNode.color[index-1][1] = 0;
            ActiveLineSetColorNode.color[index-1][2] = 0;
            ActiveLineSetColorNode.color[index][0] = 1;
            ActiveLineSetColorNode.color[index][1] = 1;
            ActiveLineSetColorNode.color[index][2] = 1;
   }
  }

   else {
        ActiveLineSetCoordinateNode.point[index] =
newPointPositionsArray[index];
        coordIndex_changed[index+lineIndex] =
coordIndex_changed[index+lineIndex-2];
        coordIndex_changed[index+lineIndex+1] = index;
        coordIndex_changed[index+lineIndex+2] = -1;
        ActiveLineSetColorNode.color[index-1][0] = 1;
        ActiveLineSetColorNode.color[index-1][1] = 0;
        ActiveLineSetColorNode.color[index-1][2] = 0;
        ActiveLineSetColorNode.color[index][0] = 1;
        ActiveLineSetColorNode.color[index][1] = 1;
        ActiveLineSetColorNode.color[index][2] = 1;
        lineIndex += 2;

   }
   index ++;
 }
}
```

The data processing is a little bit more complex regarding the ActiveLineSet *coordIndex* field, due to the fact that the syntax is special: this field specifies indices describing the path of polylines. Each polyline is distinguished among the points by adding “-1” after their indexes. Thus, every two points, the value “-1” is added in the *coordIndex_changed* array. After that, this array is routed to the coordIndex field. That explains why a few incremental variables like *Index* or *lineIndex* are used. A specific code was needed to process the first and second values (for index = 0 and index = 1).

Regarding line colors, a white color is associated with the current point; a red color is associated with points displayed before the current point. The result of this is that “old” lines come into view with a red color and the current line is drawn with graduated shading color (from red to white). In that way, it is easier to visualize what is the current line. Conceivably line colors also may be defined according to depth.

All nodes work and show the graphic result in “real time” matching data collection times by the AUV.

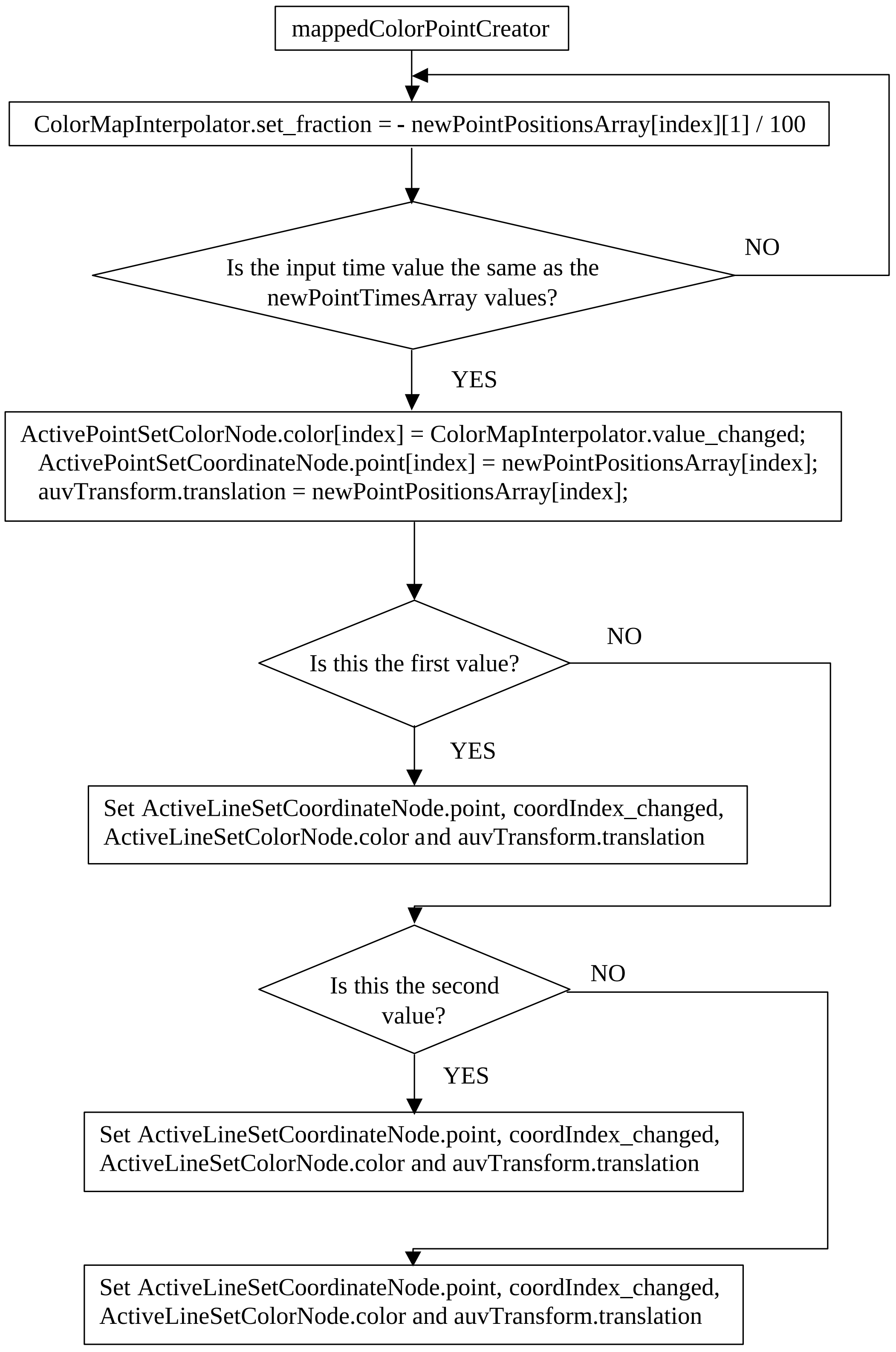

mappedColorPointCreator
ColorMapInterpolator.set_fraction = - newPointPositionsArray[index][1] / 100
Is the input time value the same as the newPointTimesArray values?
NO
YES
ActivePointSetColorNode.color[index] = ColorMapInterpolator.value_changed;
ActivePointSetCoordinateNode.point[index] = newPointPositionsArray[index];
auvTransform.translation = newPointPositionsArray[index];
Is this the first value?
NO
YES
Set ActiveLineSetCoordinateNode.point, coordIndex_changed, ActiveLineSetColorNode.color and auvTransform.translation
Is this the second value?
NO
YES
Set ActiveLineSetCoordinateNode.point, coordIndex_changed, ActiveLineSetColorNode.color and auvTransform.translation
Set ActiveLineSetCoordinateNode.point, coordIndex_changed, ActiveLineSetColorNode.color and auvTransform.translation

Diagram 3. Flowchart Execution for the Function mappedColorPointCreator().

(5) The Script Debugger. This script is useful to validate whether or not values that are put in different nodes like ActivePointSet, CompletePointSet or ActiveLineSet. User can easily check whether those nodes work properly and values make sense or not. This information is printed on the browser's VRML console. Values are sent to the debugger script via ROUTEs.

e. Outside the Prototype

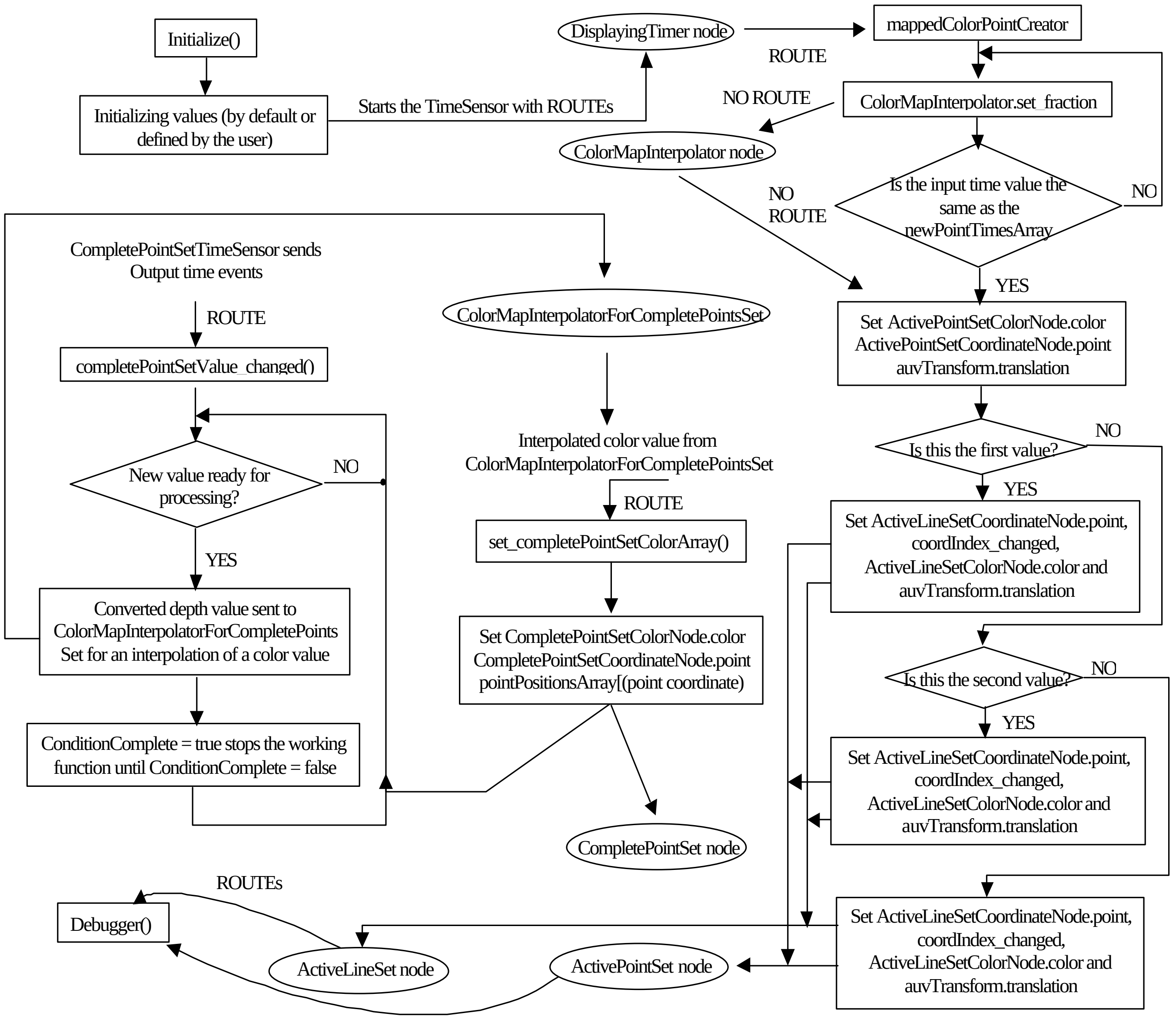
The PROTO declaration creates a complete new PROTO node by retrieving the PROTO declaration within the file where this declaration is located. All field declarations are implicit (no need to specify values unless if they have to be different from default values). By using the ExternProto Declare syntax, this prototype can be reused in external files (other VRML worlds) as many times as needed.

A TimeSensor called DisplayTimer sends time events to the function mappedColorPointCreator via ROUTEs.

This program uses few ROUTEs, i.e. the minimum required. The previous version used many more ROUTEs, which implies that more variables, buffer arrays were created. It worked but took a bit more of memory resources. It does not matter to care about saving memory if it is a question of displaying a few points (or lines). Nevertheless when an AUV operation requires hundreds of points, it becomes more interesting to find a better solution to optimize the program. That is possible by defining node variables inside the script node. They are linked to nodes declared outside the script inside the PROTO instance. Thus, it is possible to send or receive values to/from nodes without writing ROUTEs. This method avoids use of redundant buffer arrays.

f. Complete Diagram of the Program

The whole source code of the waypoint track generator is presented in Appendix B. A full diagram recapitulates how the whole program works and is shown on the next page.



Initialize()
Initializing values (by default or defined by the user)
Starts the TimeSensor with ROUTEs
DisplayingTimer node
ROUTE
mappedColorPointCreator
NO ROUTE
ColorMapInterpolator.set_fraction
ColorMapInterpolator node
NO ROUTE
Is the input time value the same as the newPointTimesArray
NO
YES
Set ActivePointSetColorNode.color ActivePointSetCoordinateNode.point auvTransform.translation
Is this the first value?
NO
YES
Set ActiveLineSetCoordinateNode.point, coordIndex_changed, ActiveLineSetColorNode.color and auvTransform.translation
Is this the second value?
NO
YES
Set ActiveLineSetCoordinateNode.point, coordIndex_changed, ActiveLineSetColorNode.color and auvTransform.translation
Set ActiveLineSetCoordinateNode.point, coordIndex_changed, ActiveLineSetColorNode.color and auvTransform.translation
CompletePointSetTimeSensor sends Output time events
ROUTE
completePointSetValue_changed()
New value ready for processing?
NO
YES
Converted depth value sent to ColorMapInterpolatorForCompletePoints Set for an interpolation of a color value
ConditionComplete = true stops the working function until ConditionComplete = false
ColorMapInterpolatorForCompletePointsSet
Interpolated color value from ColorMapInterpolatorForCompletePointsSet
ROUTE
set_completePointSetColorArray()
Set CompletePointSetColorNode.color CompletePointSetCoordinateNode.point pointPositionsArray[(point coordinate)
CompletePointSet node
ROUTEs
Debugger()
ActiveLineSet node
ActivePointSet node

Diagram 4. Complete Flowchart Diagram Execution of the Program Waypoint Track Generator.

D. MINE CONTACTS

1. Introduction

During operations, when the AUV locates a mine or mine-like contact, information about contact coordinates, contact name and contact picture is stored by the ADS software. Identification and coordinates can then be used to represent the contact object in the VRML world.

These features might be integrated in the waypoint track generator scene. Currently, there is no integrated contact visualization but this is a good step for a future work.

More script programs are needed to integrate mine contacts but it is also necessary to extend the contact/mine library by adding more 3D models (such as AUV models). The following example describe the conception of a typical 3D mine model, called Manta mine.

2. The Manta Prototype

This section describes design of a Manta mine model created with VRML, using publicly available documentation including mine pictures and dimensions.

The model is a prototype instance that can be reused in external files. Defined fields allow authors to customize the model. In this case, for the Manta mine, it is possible to change its color. About the conception of the model, it is the same way to create as the AUV model. Specifically there are many *Extrusion* Nodes, and one *IndexedFaceSet* node.

The Manta model is built from the following components:

- the main mine body, which is a cross section extruded along a circle spine in such a way that it has the same effect as a revolution.

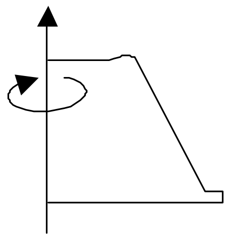


Figure 16. Manta Model Body Extrusion.

- four rings, to handle or carry the mine are designed with Extrusion nodes. The cross section is a circle and the spine is also a circle made to create a ring shape. The ring columns are made by extrusion in the same way that the main body. Using the emissiveColor field in the Material node of the shape creates the light reflection effect that gives the feeling rings are metallic.

Figure 17. Manta Model Ring Extrusion (ring + column).

- Ring supports are IndexedFaceSet nodes, with four faces.

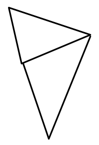


Figure 18. Manta Model Ring Support.

- Holes in the main body are black cylinders.

Here below is a picture of the 3D mine model with all its components:

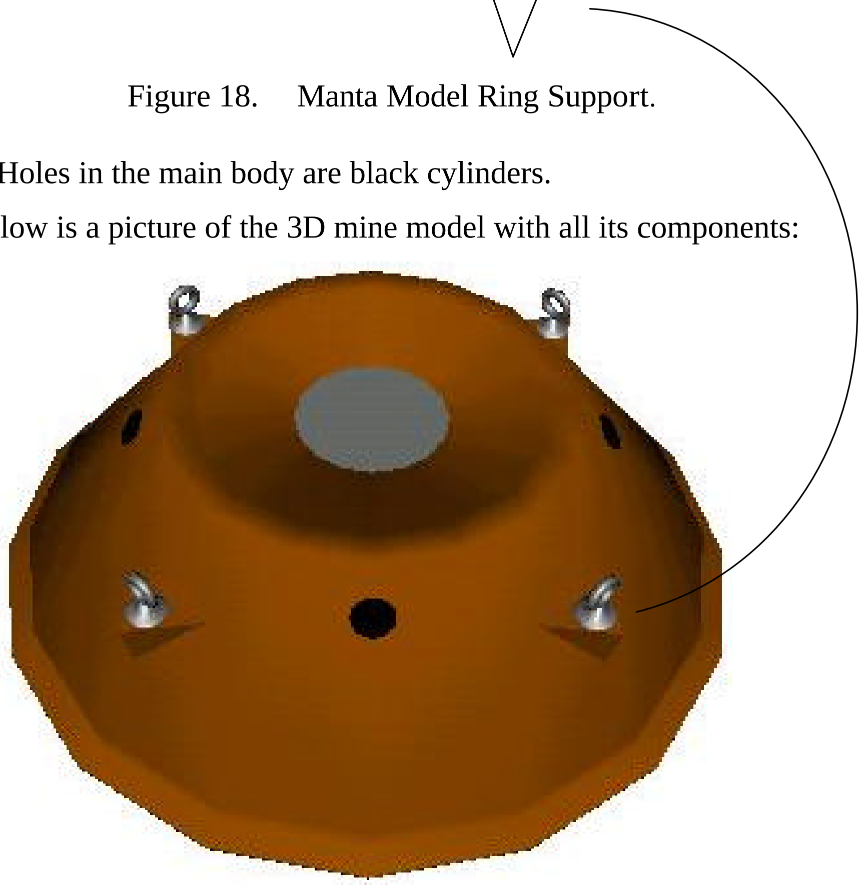


Figure 19. 3D Manta Mine Model.

The most interesting part of this model is the use of a LOD (Level Of Detail) node. This is an interesting technique to support automatic selection between lots of detail for maximum realism and quick drawing for maximum interactivity. By using the LOD node, shapes far away from the viewer need not to be drawn with as much detail as those shapes that are closer. By selecting different versions of a shape in varying levels of details, the LOD node gives a compromise between realism and interactivity. According to range values (distance between user and shapes), different versions of the shape are drawn.

These different shapes are included in the group by listing them in the level field of the node. The value of the range field specifies a list of distances (between viewer and shapes) at which the browser switches from, from one level of detail to another.

For the Manta prototype, four shapes are included in the LOD node. The more detailed shape is the mine with all its components. At 10 meters, mine rings are replaced by spherical shapes. At 50 meters, the mine is replaced by a cylinder. At 100 meters, there is no shape rendered anymore: there is only a *Worldinfo* node. This node provides comments, such as title text that could be extracted by the browser and displayed to a viewer. In effect, it serves as a null node at long distances.

Three viewpoints nodes are put in the scene at the distances where the LOD node switches the shapes. These make user navigation and author testing easier.

The whole source code of the Manta mine model is presented in Appendix C.

E. SUMMARY

This chapter provides a description of the 3D VRML/X3D AUV model in detail. Each node is described with specific explanations. Precise explanations are then provided about the development of the Waypoint Track Generator, which create a 3D virtual world for AUV mission simulations. Finally, it provides a description of the 3D VRML/X3D mine model in detail. Each node is described with specific explanations.

VI. EXPERIMENTAL RESULTS

A. EVALUATION OF OUTPUTS

Currently, the Waypoint Track Generator works satisfactorily. It is still missing some features but scenes are properly generated and displayed on screen.

This work is in accordance with the schedule of conditions drawn up and has the following features:

- From sets of point coordinates and time fractions (written inside the program body), the AUV path is drawn using color points on a VRML scene. A colored point allows to visualize the bathymetry. Points are displayed according to time stamps combined with point coordinates.
- Using only sets of point coordinates, the AUV path is drawn using color points on a VRML scene. Unlike the precedent method, colored points are displayed together, at the beginning, without any time condition.
- The last method is experimental: lines are used for drawing the AUV path, instead points. This last method was not required by the schedule of conditions but can be a good way to think about improvements and future works. The 3D AUV model can also be added for more realism.

For a maximum efficiency, the program code is written mainly with EcmaScript (JavaScript) source code, trying to reduce the number of ROUTEs. This approach is effective.

Note that color interpolators are used to convert depth values into color values. This innovation increases computing performances compared to a set of comparative conditional "if-then-else if" statements to find the right colors in a previously defined range. There is more work to do and more thoughts to have but henceforth the Waypoint Track Generator can be used for ADS.

B. SPECIFIC EXAMPLE MISSION

This chapter deals with a test that simulates AUV operations. Coordinates and time values do not come from a real mission but are based on representative examples.

The timestamped point values are put inside arrays in the example scene:

	1	2	3	4	5	6	7	8	9
coordinates	0 0 0	10 -4 0	25 -6 0	30 -8 5	38 -15 5	45 -18 5	55 -22 5	60 -25 15	60 -27 22
Time stamps	1	3	6	8	10	12	14	15	17

	10	11	12	13	14	15	16	17
coordinates	55 -30 35	48 -35 35	35 -35 35	25 -45 35	20 -55 35	15 -70 35	3 -70 35	-5 -72 40
Time stamps	18	23	28	35	37	39	43	45

	18	19	20	21	22	23	24	25
coordinates	-5 -75 50	0 -80 55	15 -75 55	30 -70 55	35 -60 55	40 -50 55	50 -34 55	65 -23 70
Time stamps	47	48	53	58	60	61	65	70

Table 1. Coordinates and Time-Stamp Values [Ref.].

If the author chooses a path made with points, result appears as in Figure VI-1.

Note that color points are bigger than in reality: they have been touched up in this picture otherwise we could not see anything. Points are not typically distinguishable with a white background. A black background fits well with small points (good contrast) but is not well suited as a report figure.

A possible task for future work is to use a different type of geometry, perhaps billboarded to always face the user, as a way to overcome some of the visualization difficulties inherent in fixed single-pixel-size points and single-pixel-width lines.

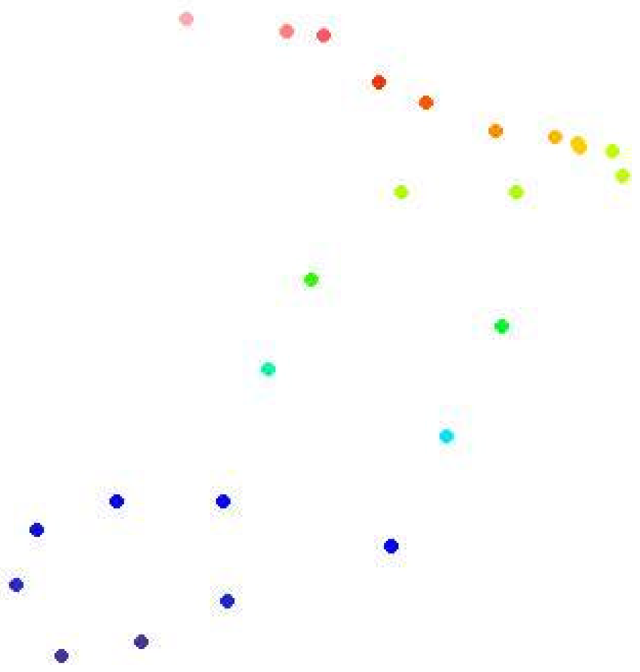


Figure 20. AUV Path Made with Points using VRML. Point images have been augmented to improve default contrast.

If a path made with lines is chosen instead, the result appears like in the picture below:

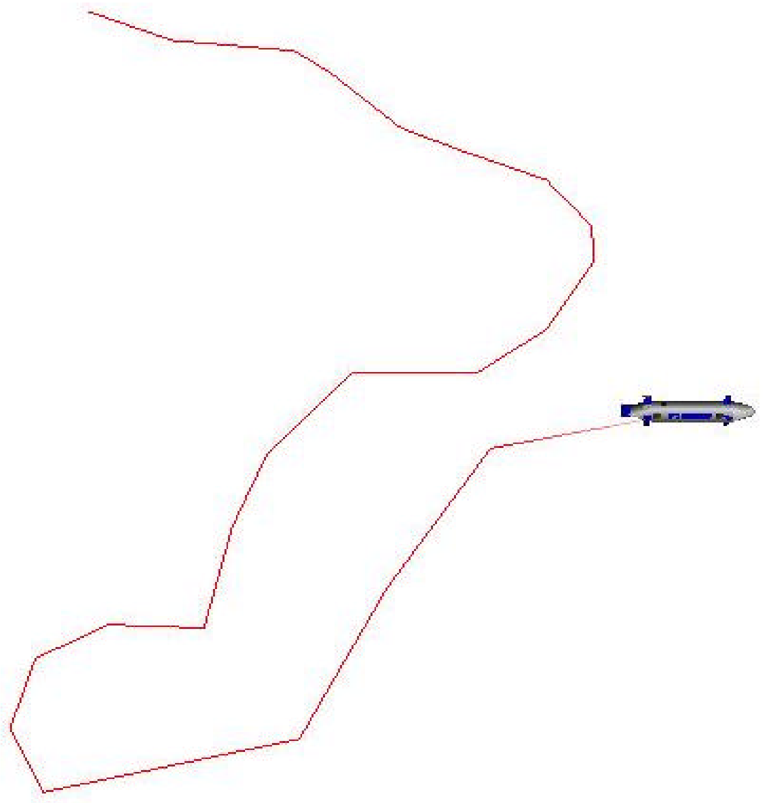


Figure 21. AUV Path Made with Lines using VRML.

C. SUMMARY

This chapter discusses about evaluate outputs of the Waypoint Track Generator and provides some advices for future work. This chapter also deals with an exemplary VRML/X3D scene representative of AUV missions.

VII. CONCLUSIONS AND FUTURE WORK

A. CONCLUSIONS

The main purpose of this project is to create a simple scene based on VRML. From coordinate data and time data, generated by the ADS software, a 3D scene is built to represent the path that followed the AUV during operations. After constructing exemplary VRML scenes, these prototypes will be integrated into ADS in the future to produce mission archive set. Automatically creating visual mission archives to help visualize and easily understand what the AUV did during operations will be a new achievement.

The result is a VRML scene, principally constructed at load time with JavaScript codes, which allows the AUV path to be displayed in a 3D world according three modes:

- AUV path is displayed with a large quantity of points that represent intermediate coordinate points. Each point gets his own color, which symbolized the depth of the AUV at this point. All the points are displayed at the same time, without caring about the time reference list.
- AUV path is displayed with a quantity of points that are coordinate points, but each point is associated with time stamps. The bathymetry is symbolized by colors but points appear according to their respective time stamps.
- AUV path is shown with line segments with a single color: there is no color representation of the bathymetry. The lines are drawn according to their respective time stamps. 3D AUV models can be added to this scene.

B. RECOMMENDATIONS FOR FUTURE WORK

This work achieves the goals originally set. As with any success, new lessons are learned and new challenges provide further opportunities.

Firstly, currently, the display mode has to be defined inside the VRML file, before starting it. It may be better if the display mode could directly be chosen after starting the scene. This choice could be made by creating three Text nodes with mode names. When user would click on one of them, display mode would be defined.

Secondly, durationActivePoint and timeLatestActivePoint values (range of values displayed on screen) cannot be modified when the scene runs. A good idea is to create sliders (combination of cylinder and sphere shapes) that could decrease/increase durationActivePoint and timeLatestActivePoint values in real time.

When AUV operations are represented by a line path, a 3D AUV model is added and moves following the path. Nevertheless its movement is jerky. A good improvement would be to interpolate AUV motion to make it smoother and more realistic.

Finally, mine-like contact coordinates have to be represented within the VRML scene using the same way as AUV coordinates. 3D mine models would be included in the scene, located at the coordinates provided by ADS data. More AUV and mine models might also be created to feed the 3D library.

This project is a preliminary work, which deserves to be continued. It lays the foundation for a variety of future works that can become a useful tool contributing towards NPS AUV research.

APPENDIX A. ARIES AUV X3D MODEL

3D ARIES AUV model source code (VRML):

```
#VRML V2.0 utf8
# X3D-to-VRML-97 XSL translation autogenerated by X3dToVrml97.xsl
# http://www.web3D.org/TaskGroups/x3d/translation/X3dToVrml97.xsl
# [X3D]
# [Scene]

NavigationInfo {
  type [ "EXAMINE" "ANY" ]
}
Viewpoint {
  description "Entry"
  position -0.05 0 2
}
DEF auv Group {
  children [
  Transform {
    translation 0.6223 0.13335 0
    children [
    DEF A_Plane Shape {
      appearance Appearance {
        material Material {
          diffuseColor 0 0 1
        }
      }
      geometry IndexedFaceSet {
        coordIndex [ 0 3 2 1 -1 4 5 6 7 -1 0 1 5 4 -1 1 2 6 5 -1 2
3 7 6 -1 0 3 7 4 -1 ]
        creaseAngle 3.14159
        coord Coordinate {
          point [ 0.0635, 0, 0.0127, 0.0381, 0.1778, -0.0127, -0.0381, 0.1778,
- 0.0127, -0.0889, 0, -0.0127, 0.0635, 0, 0.0127, 0.0381, 0.1778, 0.0127, -.0381,
0.1778, 0.0127, -0.0889, 0, 0.0127 ]
        }
      }
    }
```

```
          ]
        }
        Transform {
          translation -0.7747 0.13335 0
          children [
           USE A_Plane
           Transform {
            translation 0 0.1778 0
            children [
            Shape {
              appearance Appearance {
                material Material {
                  diffuseColor 1 0.3 0
                }
              }
              geometry IndexedFaceSet {
                coordIndex [ 0 9 10 -1, 0 10 1 -1, 2 1 10 -1, 3 2 10 -1, 4 3 10 -1, 5
4 10 -1, 6 5 10 -1, 7 6 10 -1, 8 7 10 -1, 9 8 10 -1, 0 1 2 3 4 5 6 7 8 9 -1, ]
                creaseAngle 1.57
                coord Coordinate {
                  point [ -0.1 0 0, 0 0 -0.05,  0.019 0 -0.046, 0.0355 0 -0.0355,
0.046 0 -0.019, 0.05 0 0, 0.046 0 0.019, 0.0355 0 0.0355, 0.019 0 0.046, 0 0 0.05, 0
0.03 0 ]
                }
              }
            }
            ]
          }
          ]
        }
        Transform {
          rotation 1 0 0 3.14159267
          translation 0.6223 -0.13335 0
          children [
          USE A_Plane
          ]
        }
        Transform {
          rotation 1 0 0 3.14159267
          translation -0.7747 -0.13335 0
          children [
          USE A_Plane
          ]
        }
        Transform {
          rotation 1 0 0 1.5708
          translation -0.7747 0 0.20955
          children [
          USE A_Plane
          ]
        }
        Transform {
          rotation 1 0 0 1.5708
          translation 0.6223 0 0.20955
          children [
          USE A_Plane
          ]
        }
        Transform {
          rotation 1 0 0 -1.5708
          translation 0.6223 0 -0.20955
          children [
          USE A_Plane
          ]
        }
        Transform {
          rotation 1 0 0 -1.5708
          translation -0.7747 0 -0.20955
          children [
          USE A_Plane
          ]
        }
        Transform {
          translation -0.4953 0 0
          children [
          Shape {
            appearance Appearance {
              material Material {
                diffuseColor 0.2 0.2 0.2
              }
            }
            geometry Cylinder {
              height 0.29
              radius 0.0635
            }
          }
          ]
        }
        Transform {
          rotation 1 0 0 1.5708
          translation 0.4699 0 0
          children [
          Shape {
            appearance Appearance {
              material Material {
                diffuseColor 0.2 0.2 0.2
              }
            }
            geometry Cylinder {
              height 0.44
              radius 0.0635
            }
          }
          ]
        }
        Transform {
          rotation 1 0 0 1.5708
          translation -0.6223 0 0
          children [
          Shape {
            appearance Appearance {
              material Material {
                diffuseColor 0.2 0.2 0.2
              }
            }
            geometry Cylinder {
              height 0.44
              radius 0.0635
            }
          }
          ]
```

```
          }
          Group {
            children [
            Shape {
              appearance Appearance {
                material Material {
                  diffuseColor 0.9 0.9 0.9
                }
              }
              geometry IndexedFaceSet {
                coordIndex [ 0 26 34 33 32 31 25 1 –1 1 25 29 28 2 –1 2 28 35 36 37 38
27 3 –1 0 3 27 30 26 –1 0 4 1 –1 0 1 4 –1 1 5 2 –1 1 2 5 –1 2 6 3 –1 2 3 6 –1 3 7 0 –1
3 0 7 –1 7 0 8 –1 7 8 0 –1 8 0 9 –1 8 9 0 –1 9 0 4 –1 9 4 0 –1 4 1 10 –1 4 10 1 –1 10
1 –1 10 11 1 –1 11 1 5 –1 11 5 1 –1 5 2 12 –1 5 12 2 –1 12 2 13 –1 12 13 2 –1 13 2 6 -
13 6 2 –1 6 3 14 –1 6 14 3 –1 14 3 15 –1 14 15 3 –1 15 3 7 –1 15 7 3 –1 4 10 16 –1 10
1 16 –1 11 5 17 –1 5 12 18 –1 12 13 19 –1 13 6 19 –1 6 14 20 –1 14 15 20 –1 15 7 21 –1
7 8 22 –1 8 9 23 –1 9 4 23 –1 4 16 23 –1 11 17 16 –1 5 18 17 –1 12 19 18 –1 6 20 19 –1
20 15 21 -1  21 7 22 –1 22 8 23 –1 23 16 24 –1 16 17 24 –1 17 18 24 –1 18 19 24 –1 19
20 24 –1  20 21 24 –1 21 22 24 –1 22 23 24 –1 26 27 30 –1 25 26 30 29 –1 25 29 28 –1
27 28 29 30 –1 31 32 36 35 –1 32 33 37 36 –1 34 38 37 33 -1 ]
                creaseAngle 3.14159
                coord Coordinate {
                  point [ 0.6985, 0.13335, -0.20955, 0.6985, 0.13335, 0.20955, 0.6985,
-0.13335, 0.20955, 0.6985, -0.13335, -0.20955, 1.05, 0.085, 0, 1.05, 0, 0.1143, 1.05,
- 0.085, 0, 1.05, 0, -0.1143, 1.05, 0.04572, -0.098985, 1.05, 0.079188, -0.05715,
1.05, .079188, 0.05715, 1.05, 0.04572, 0.098985, 1.05, -0.04572, 0.098985, 1.05, -
0.079188, .05715, 1.05, -0.079188, -0.05715, 1.05, -0.04572, -0.098985, 1.1, 0.04064,
0.02032, 1.1, 0.02032, 0.06096, 1.1, -0.02032, 0.06096, 1.1, -0.04064, 0.02032, 1.1, -
0.04064, 0.02032, 1.1, -0.02032, -0.06096, 1.1, 0.02032, -0.06096, 1.1, 0.04064, -
0.02032, 1.11, 0, 0, -0.6985, 0.13335, 0.20955,  -0.6985, 0.13335, -0.20955, -0.6985,
-0.13335,  -0.20955, -0.6985, -0.13335, 0.20955, -1.1303, 0, 0.20955, -1.1303, 0, -
0.20955, -0.6985, 0.13335, 0.0635, -0.8509, 0.13335, 0.0635, -0.8509, 0.13335, -
0.0635, -0.6985, .13335, -0.0635, -0.6985, -0.13335, 0.0635, -0.8509, -0.13335,
0.0635, -0.8509, -0.13335, -0.0635, -0.6985, -0.13335, -0.0635, ]
                }
              }
            }
            ]
          }
          Transform {
            translation -1.1557 0 0.09525
            children [
            Group {
              children [
              DEF Stbd_Blade Group {
                children [
                Transform {
```

```
                rotation 0 1 0 -0.39
                children [
                Shape {
                  appearance Appearance {
                    material Material {
                      diffuseColor 0 0 1
                    }
                  }
                  geometry IndexedFaceSet {
                    coordIndex [ 0 1 2 3 4 5 6 7 –1 0 7 6 5 4 3 2 1 -1 ]
                    coord Coordinate {
                       point [ 0, 0,  -0.00508, 0, 0.02540,  -0.02032, 0, 0.04572,  -
0.01524, 0, 0.05080,  -0.00508, 0, 0.05080, 0.00508, 0, 0.04572, 0.01524, 0, 0.02540,
.02032, 0, 0, 0.00508 ]
                    }
                  }
                }
                ]
              }
              ]
            }
            Transform {
              rotation 1 0 0 1.5708
              children [
              USE Stbd_Blade
              ]
            }
            Transform {
              rotation 1 0 0 3.14159267
              children [
              USE Stbd_Blade
              ]
            }
            Transform {
              rotation 1 0 0 -1.5708
              children [
              USE Stbd_Blade
              ]
            }
            Transform {
              rotation 0 0 1 1.5708
              translation 0.0281 0 0
              children [
              Shape {
                appearance Appearance {
                  material Material {
                    diffuseColor 0 0 1
                  }
                }
                geometry Cylinder {
                  height 0.0762
                  radius 0.008
                }
              }
              ]
            }
            Transform {
              rotation 0 0 1 1.5708
              translation -0.015 0 0
              children [
              Shape {
                appearance Appearance {
                  material Material {
                    diffuseColor 0 0 1
                  }
                }
                geometry Cone {
                  bottomRadius 0.008
                  height 0.01
                }
              }
              ]
            }
            Shape {
              geometry Extrusion {
                beginCap FALSE
                creaseAngle 2
                 crossSection [  1.00  0.00,   0.92 -0.38,  0.71 -0.71,   0.38 -0.92,
0.00 -1.00,  -0.38 -0.92, -0.71 -0.71,  -0.92 -0.38, -1.00 -0.00,  -0.92  0.38, -0.71
0.71,  -0.38  0.92,  0.00  1.00,   0.38  0.92,  0.71  0.71,   0.92  0.38,  1.00  0.00
]
                endCap FALSE
                scale [ 0.08 0.08, 0.07 0.07, 0.06 0.06, 0.07 0.07, 0.08 0.08 ]
                spine [ -0.08 0 0, 0.08 0 0, 0.08 0 0, -0.08 0 0, -0.08 0 0 ]
              }
              appearance Appearance {
                material Material {
                  diffuseColor 0 0 1
                }
              }
            }
            ]
          }
          ]
        }
        Transform {
          translation -1.1557 0 -0.09525
          children [
          Group {
            children [
            DEF Port_blade Group {
              children [
              Transform {
                rotation 0 1 0 0.39
                children [
                Shape {
                  appearance Appearance {
                    material Material {
                      diffuseColor 0 0 1
                    }
                  }
                  geometry IndexedFaceSet {
                    coordIndex [ 0 1 2 3 4 5 6 7 –1 0 7 6 5 4 3 2 1 -1 ]
                    coord Coordinate {
                       point [ 0, 0, -0.00508, 0, 0.02540, -0.02032, 0, 0.04572, -
0.01524, 0, 0.05080, -0.00508, 0, 0.05080, 0.00508, 0, 0.04572, 0.01524, 0, 0.02540,
0.02032, 0, 0, 0.00508 ]
                    }
                  }
                }
                ]
              }
              ]
```

```
}
Transform {
  rotation 1 0 0 1.5708
  children [
  USE Port_blade
  ]
}
Transform {
  rotation 1 0 0 3.14159267
  children [
  USE Port_blade
  ]
}
Transform {
  rotation 1 0 0 -1.5708
  children [
  USE Port_blade
  ]
}
Transform {
  rotation 0 0 1 1.5708
  translation 0.0281 0 0
  children [
  Shape {
    appearance Appearance {
      material Material {
        diffuseColor 0 0 1
      }
    }
    geometry Cylinder {
      height 0.0762
      radius 0.008
    }
  }
  ]
}
Transform {
  rotation 0 0 1 1.5708
  translation -0.015 0 0
  children [
              Shape {
                appearance Appearance {
                  material Material {
                    diffuseColor 0 0 1
                  }
                }
                geometry Cone {
                  bottomRadius 0.008
                  height 0.01
                }
              }
              ]
            }
            Shape {
              geometry Extrusion {
                beginCap FALSE
                creaseAngle 2
                 crossSection [1.00  0.00, 0.92 -0.38,   0.71 -0.71,    0.38 -0.92,
0.00 -1.00,  -0.38 -0.92, -0.71 -0.71,  -0.92 -0.38, -1.00 -0.00,  -0.92  0.38, -0.71
0.71,  -0.38  0.92,  0.00  1.00,   0.38  0.92,  0.71  0.71,   0.92  0.38,  1.00  0.00]
                endCap FALSE
                scale [ 0.08 0.08, 0.07 0.07, 0.06 0.06, 0.07 0.07, 0.08 0.08 ]
                spine [ -0.08 0 0, 0.08 0 0, 0.08 0 0, -0.08 0 0, -0.08 0 0 ]
              }
              appearance Appearance {
                material Material {
                  diffuseColor 0 0 1
                }
              }
            }
            ]
          }
          ]
        }
        DEF logo Group {
          children [
          Transform {
            translation -0.475 -0.05 0.21
            children [
            Shape {
              appearance Appearance {
```

```
      material Material {
        diffuseColor 0 0 0.8
      }
    }
    geometry Text {
      string [ "NPS" ]
      fontStyle FontStyle {
        family [ "SANS" ]
        size 0.15
        style "BOLD"
      }
    }
  }
  ]
}
Transform {
  translation -0.175 -0.05 0.21
  children [
  DEF line Shape {
    appearance Appearance {
      material Material {
        diffuseColor 0 0 0.8
      }
    }
    geometry IndexedFaceSet {
      coordIndex [ 0 1 2 3 -1 ]
      solid FALSE
      coord Coordinate {
        point [ 0 0 0, 0.5 0 0, 0.5 0.02 0, 0 0.02 0, ]
      }
    }
  }
  ]
}
Transform {
  translation -0.175 -0.015 0.21
  children [
  USE line
  ]
}
```

```
      Transform {
        translation -0.175 0.02 0.21
        children [
        USE line
        ]
      }
      ]
    }
    Transform {
      rotation 0 1 0 3.14159267
      translation -0.15 0 0
      children [
      USE logo
      ]
    }
    ]
  }
```

APPENDIX B. WAYPOINT FRENCH GENERATOR X3D MODEL

VRML source code of the latest version of the waypoint track generator program:

```
	#VRML V2.0 utf8
	# X3D-to-VRML-97 XSL translation autogenerated by X3dToVrml97.xsl
	# http://www.web3D.org/TaskGroups/x3d/translation/X3dToVrml97.xsl
	# [X3D]
	# [Header]
	# [meta] filename: PointTrackGeneratorPrototype.xml
	# [meta] description: Generator of randomized colored points using script
nodes. The data arrays for coordinates and colors are generated in realtime or
everyting is displayed, depending on your choice.
	# [meta] author: Frederic Roussille
	# [meta] created: 14 May 2001
	# [meta] revised: 06 June 2001
	# [meta] url:
http://web.nps.navy.mil/~brutzman/vrml/examples/NpsMilitaryModels/Tools/Animation/Poin
tGeneratorTrack.xml
	# [meta] generator: X3D-Edit,
http://www.web3D.org/TaskGroups/x3d/translation/README.X3D-Edit.html
	# [Scene]
	PROTO PointTrackGenerator [
	  # Point coordinates in meters, referenced to local coordinate system origin
	  # Point times in seconds for local exercise clock. (Each time is clock time
in seconds, not in interval durations.).
	  # Both points and times are initially provided as a full set of values.
	  field        MFVec3f     pointPositionsArray [0 0 0, 10 -4 0, 25 -6 0, 30 -8
5, 38 -15 5, 45 -18 5, 55 -22 5, 60 -25 15, 60 -27 22, 55 -30 35, 48 -35 35, 35 -35
35, 25 -45 35, 20 -55 35, 15 -70 35, 3 -70 35, -5 -72 40, -5 -75 50, 0 -80 55, 15 -75
55, 30 -70 55, 35 -60 55, 40 -50 55, 50 -34 55, 65 -23 70] # IS
DrawPointScript.pointPositionsArray
	  field        MFTime      pointTimesArray [1, 3, 6, 8, 10, 12, 14, 15, 17, 18,
23, 28, 35, 37, 39, 43, 45, 47, 48, 53, 58, 60, 61, 65, 70] # IS
DrawPointScript.pointTimesArray
	  # totalDuration is derived from the pointTimesArray, and used to set
cycleInterval on a controlling TimeSensor clock outside the PointTrackGenerator
ProtoInstance.
	  eventOut     SFTime      totalDuration # IS DrawPointScript.totalDuration
	  # displayPointsMode settings: -1=none, 0=some (active interval), 1=all.
	  exposedField SFInt32     displayPointsMode 2 # IS
PointsGeometrySwitch.whichChoice
	  # durationActivePoints is in seconds, default initialization value is
totalDuration
```

```
        eventIn      SFTime      durationActivePoints # IS
DrawPointScript.durationActivePoints
        # timeLatestActivePoint is in seconds, default initialization value is final
point time
        eventIn      SFTime      timeLatestActivePoint # IS
DrawPointScript.timeLatestActivePoint
        eventOut     SFTime      getStartTime # IS DrawPointScript.getStartTime
        eventOut     SFTime      getStopTime # IS DrawPointScript.getStopTime
        eventIn      SFTime      mappedColorPointCreator # IS
DrawPointScript.mappedColorPointCreator
        exposedField MFString    auvName [
      "auv_ax_xml.wrl"
      ] # IS auvName.url
      ] {
        Group {
          children [
          DEF PointsGeometrySwitch Switch {
            whichChoice IS displayPointsMode
            choice [
            Shape {
              geometry DEF ActivePointSet PointSet {
                coord DEF ActivePointSetCoordinateNode Coordinate {
                }
                color DEF ActivePointSetColorNode Color {
                }
              }
            }
            Shape {
              geometry DEF CompletePointSet PointSet {
                coord DEF CompletePointSetCoordinateNode Coordinate {
                }
                color DEF CompletePointSetColorNode Color {
                }
              }
            }
            Group {
              children [
              Shape {
                geometry DEF ActiveLineSet IndexedLineSet {
                  coord DEF ActiveLineSetCoordinateNode Coordinate {
                  }
                  color DEF ActiveLineSetColorNode Color {
                  }
                }
              }
              DEF auvTransform Transform {
                scale 4 4 4
                children [
                DEF auvName Inline {
                  url IS auvName
                }
                ]
              }
              ]
            }
            ]
          }
          DEF ColorMapInterpolator ColorInterpolator {
            key [ 0, 0.12, 0.48, 0.7, 1 ]
            keyValue [ 1 1 1, 1 0 0, 0 1 0, 0 0 1, 0 0 0 ]
          }
          DEF ColorMapInterpolatorForCompletePointsSet ColorInterpolator {
            key [ 0, 0.12, 0.48, 0.7, 1 ]
            keyValue [ 1 1 1, 1 0 0, 0 1 0, 0 0 1, 0 0 0 ]
          }
          DEF CompletePointSetTimeSensor TimeSensor {
            cycleInterval 0.01
            loop TRUE
          }
          DEF DrawPointScript Script {
            # For proper operation, first insert newPoint and then newPointTimeStamp
            field        MFVec3f pointPositionsArray IS pointPositionsArray
            field        MFTime  pointTimesArray IS pointTimesArray
            field        MFVec3f newPointPositionsArray [ ]
            field        MFTime  newPointTimesArray [ ]
            field SFInt32 lineIndex 1
            eventIn      SFTime  mappedColorPointCreator IS mappedColorPointCreator
            field        SFInt32 index 0
            field        SFInt32 completeIndex 0
            eventOut     SFBool  ConditionComplete
```

```
             field        SFNode     ActivePointSetCoordinateNode USE
ActivePointSetCoordinateNode
             field        SFNode     ActivePointSetColorNode USE
ActivePointSetColorNode
             field        SFNode     CompletePointSetCoordinateNode USE
CompletePointSetCoordinateNode
             field        SFNode     CompletePointSetColorNode USE
CompletePointSetColorNode
             field        SFNode     ColorMapInterpolator USE ColorMapInterpolator
             field        SFNode     ColorMapInterpolatorForCompletePointsSet USE
ColorMapInterpolatorForCompletePointsSet
             field        SFNode     ActiveLineSetCoordinateNode USE
ActiveLineSetCoordinateNode
             field        SFNode     ActiveLineSetColorNode USE ActiveLineSetColorNode
             field        SFNode     auvTransform USE auvTransform
             eventOut     SFTime  totalDuration IS totalDuration
             eventOut     SFTime  getStartTime IS getStartTime
             eventOut     SFTime  getStopTime IS getStopTime
             eventOut     MFInt32 coordIndex_changed
             eventIn      SFTime  durationActivePoints IS durationActivePoints
             eventIn      SFTime  timeLatestActivePoint IS timeLatestActivePoint
             eventIn      SFTime  completePointSetValue_changed
             eventIn      SFColor set_completePointSetColorArray
       url [ "javascript:

       function initialize() {
        totalDuration = pointTimesArray[pointTimesArray.length-1];
        var today = new Date();
        getStartTime = Math.round(today.getTime() / 1000);
        getStopTime = getStartTime + totalDuration;
        var m = 1;

        //default values for durationActivePoint and timeLatestActivePoint
        durationActivePoint = totalDuration;
        timeLatestActivePoint = pointTimesArray[pointTimesArray.length-1];

        if(timeLatestActivePoint == durationActivePoint) {
         newPointTimesArray = pointTimesArray;
         newPointPositionsArray = pointPositionsArray;
         print('newPointTimesArray = ' + newPointTimesArray);
        }
        if(timeLatestActivePoint > durationActivePoint) {
         var firstTime = latestTime = k = 0;
         while((timeLatestActivePoint - durationActivePoint) !=
pointTimesArray[firstTime]) {
          firstTime++;
         }
         while(timeLatestActivePoint != pointTimesArray[latestTime]) {
          latestTime++;
         }
         for(var j = firstTime ; j <= latestTime ; j++) {
          newPointTimesArray[k] = pointTimesArray[j] - pointTimesArray[firstTime] + 1;
          newPointPositionsArray[k] = pointPositionsArray[j];
          k++;
         }
         print('newPointTimesArray = ' + newPointTimesArray);
        }
        if(timeLatestActivePoint < durationActivePoint) {
         print('Fatal error : timeLatestActivePoint < durationActivePoint !');
        }
        ConditionComplete = false;
       }


       function completePointSetValue_changed() {
        if(ConditionComplete == false && completeIndex <= (pointPositionsArray.length-
1)) {
         ColorMapInterpolatorForCompletePointsSet.set_fraction = -
pointPositionsArray[completeIndex][1] / 100;

print('ColorMapInterpolatorForCompletePointsSet.set_fraction['+completeIndex+'] = ' +
ColorMapInterpolatorForCompletePointsSet.set_fraction);
         //need to initialize ColorMapInterpolator.set_fraction with the first point
color otherwise the value is shifted
         ConditionComplete = true;
        }
       }


       function set_completePointSetColorArray(Value) {
         CompletePointSetColorNode.color[completeIndex] = Value;
         CompletePointSetCoordinateNode.point[completeIndex] =
pointPositionsArray[completeIndex];
         completeIndex++;
         ConditionComplete = false;
       }
```

```
        function mappedColorPointCreator(fractionValue) {
          ColorMapInterpolator.set_fraction = - newPointPositionsArray[index][1] / 100;
          //need to initialize ColorMapInterpolator.set_fraction with the first point
color otherwise the value is shifted
          if(Math.floor(fractionValue) == (newPointTimesArray[index] + getStartTime)) {
           ActivePointSetColorNode.color[index] = ColorMapInterpolator.value_changed;
           ActivePointSetCoordinateNode.point[index] = newPointPositionsArray[index];
           auvTransform.translation = newPointPositionsArray[index];
           if(index <= 1) {
                  ActiveLineSetCoordinateNode.point[index] =
newPointPositionsArray[index];
               coordIndex_changed[index] = index;
                  ActiveLineSetColorNode.color[index][0] = 1;
                  ActiveLineSetColorNode.color[index][1] = 1;
                  ActiveLineSetColorNode.color[index][2] = 1;
                  auvTransform.translation = newPointPositionsArray[index];
                  if(index == 1) {
                           ActiveLineSetCoordinateNode.point[index] =
newPointPositionsArray[index];
                    coordIndex_changed[index] = index;
                    coordIndex_changed[index+1] = -1;
                       ActiveLineSetColorNode.color[index-1][0] = 1;
                       ActiveLineSetColorNode.color[index-1][1] = 0;
                       ActiveLineSetColorNode.color[index-1][2] = 0;
                       ActiveLineSetColorNode.color[index][0] = 1;
                       ActiveLineSetColorNode.color[index][1] = 1;
                       ActiveLineSetColorNode.color[index][2] = 1;
                  }
           }
           else {
                  ActiveLineSetCoordinateNode.point[index] =
newPointPositionsArray[index];
                  coordIndex_changed[index+lineIndex] =
coordIndex_changed[index+lineIndex-2];
                  coordIndex_changed[index+lineIndex+1] = index;
                  coordIndex_changed[index+lineIndex+2] = -1;
                  ActiveLineSetColorNode.color[index-1][0] = 1;
                  ActiveLineSetColorNode.color[index-1][1] = 0;
                  ActiveLineSetColorNode.color[index-1][2] = 0;
                  ActiveLineSetColorNode.color[index][0] = 1;
                  ActiveLineSetColorNode.color[index][1] = 1;
```

```
                ActiveLineSetColorNode.color[index][2] = 1;
                lineIndex += 2;
          }
          //print('ActivePointSetCoordinateNode.point[' +index +'][0]=' +
ActivePointSetCoordinateNode.point[index][0]);
          //print('ActivePointSetCoordinateNode.point[' +index +'][1]=' +
ActivePointSetCoordinateNode.point[index][1]);
          //print('ActivePointSetCoordinateNode.point[' +index +'][2]=' +
ActivePointSetCoordinateNode.point[index][2]);
          index ++;
        }
       }
       " ]
           }
           DEF Debugger Script {
             eventIn      MFVec3f set_debugcoordinate
             eventIn      MFColor set_debugcolor
             eventIn      MFVec3f set_debugcoordinateC
             eventIn      MFColor set_debugcolorC
             eventIn      MFVec3f set_debugcoord
             eventIn      MFInt32 set_debugcoordIndex_changed
       url [ "javascript:
       function set_debugcoordinate(Value) {
        print('ActivePointSet : CoordinatePointArrray = ' + Value);
       }

       function set_debugcolor(Valeur) {
        print('ActivePointSet : ColorPointArray = ' + Valeur);
       }

       function set_debugcoordinateC(Value) {
        print('CompletePointSet : CoordinatePointArrray = ' + Value);
        print(' ');
       }

       function set_debugcolorC(Valeur) {
        print('CompletePointSet : ColorPointArray = ' + Valeur);
       }

       function set_debugcoord(Valeur) {
        print('ActiveLineSet : Coordinate.point = ' + Valeur);
```

```
         print(' ');
        }

        function set_debugcoordIndex_changed(Valeur) {
         print('DrawPointScript : coordIndex_changed = ' + Valeur);
        }
        " ]
            }
            ]
          }
          ROUTE CompletePointSetTimeSensor.cycleTime TO
DrawPointScript.completePointSetValue_changed
          ROUTE ColorMapInterpolatorForCompletePointsSet.value_changed TO
DrawPointScript.set_completePointSetColorArray
          ROUTE DrawPointScript.coordIndex_changed TO ActiveLineSet.set_coordIndex
          ROUTE ActivePointSetCoordinateNode.point_changed TO
Debugger.set_debugcoordinate
          ROUTE ActivePointSetColorNode.color_changed TO Debugger.set_debugcolor
          ROUTE ActiveLineSetCoordinateNode.point TO Debugger.set_debugcoord
          ROUTE DrawPointScript.coordIndex_changed TO
Debugger.set_debugcoordIndex_changed
          ROUTE CompletePointSetCoordinateNode.point_changed TO
Debugger.set_debugcoordinateC
          ROUTE CompletePointSetColorNode.color_changed TO Debugger.set_debugcolorC
        }
        # Example scene goes here
        NavigationInfo {
          type [ "EXAMINE" "ANY" ]
        }
        Viewpoint {
          description "MainView"
          position 0 -50 200
        }
        DEF TrackGeneratorInstance PointTrackGenerator {
        }
        DEF DisplayingTimer TimeSensor {
        }
        ROUTE TrackGeneratorInstance.getStartTime TO DisplayingTimer.set_startTime
        ROUTE TrackGeneratorInstance.getStopTime TO DisplayingTimer.set_stopTime
        ROUTE TrackGeneratorInstance.totalDuration TO DisplayingTimer.set_cycleInterval
        ROUTE DisplayingTimer.time_changed TO
TrackGeneratorInstance.mappedColorPointCreator
```

APPENDIX C. MANTA UNDERWATER MINE X3D MODEL

3D Manta mine model source code (VRML):

```
#VRML V2.0 utf8
# X3D-to-VRML-97 XSL translation autogenerated by X3dToVrml97.xsl
# http://www.web3D.org/TaskGroups/x3d/translation/X3dToVrml97.xsl
# [X3D]
# [Header]
# [meta] filename: MantaPrototype.xml
# [meta] description: Italian Manta bottom mine, with truncated cone and
handling padeyes.
# [meta] author: Frederic Roussille
# [meta] created: 8 May 2001
# [meta] revised: 16 May 2001
# [meta] url:
http://www.web3D.org/TaskGroups/x3d/translation/examples/NpsMilitaryModels/Weapons/Und
erwaterMines/MantaPrototype.xml
# [meta] photo: http://www.cisatlantic.com/trimix/strike/minelocator.jpg
# [meta] photo: http://www.cisatlantic.com/trimix/strike/Mine1.jpg
# [meta] photo: http://www.cisatlantic.com/trimix/strike/Mine2.jpg
# [meta] photo: http://www.fas.org/man/dod-101/navy/docs/swos/cmd/miw/Sp6-4-
1/sld055.htm
# [meta] generator: X3D-Edit,
http://www.web3D.org/TaskGroups/x3d/translation/README.X3D-Edit.html
# [Scene]


PROTO MantaMine [
  exposedField SFColor     MineColor 0.6 0.3 0 # IS MineColor.diffuseColor
  field        SFString    viewpointDescription "Manta mine" # IS
EntryViewpoint.description
] {
  # Bad CosmoPlayer bug: only first node is used in Prototype. Thus we wrap
everything inside a Group. Beurk (bleah)!!
  Group {
    children [
    DEF EntryViewpoint Viewpoint {
      description IS viewpointDescription
      orientation 1 0 0 -0.4
      position 0 1 3
    }
```

```
          LOD {
            range [ 10 50 100 ]
            level [
            Group {
              children [
              Viewpoint {
                description "Manta  top view"
                orientation 1 0 0 -1.57
                position 0 2 0
              }
              Viewpoint {
                description "Manta  side view"
                orientation 0 1 0 -1.57
                position -2 0 0
              }
              Transform {
                rotation 1 0 0 1.57
                scale 1.5 1.5 1
                children [
                Shape {
                  appearance Appearance {
                    material DEF MineColor Material {
                      diffuseColor IS MineColor
                    }
                  }
                  geometry Extrusion {
                    beginCap FALSE
                    creaseAngle 157
                    crossSection [ 0.1 0, 0.22 -0.01, 0.2275 -0.05 0.2675 -0.05, 0.49
0.4, 0.49 0.47, 0.53 0.47, 0.53 0.48, 0 0.48, 0 0.22, 0.0675 0.22, 0.1 0, ]
                    endCap FALSE
                    spine [ 0.001 0 0, 0.00092 -0.00038 0, 0.00071 -0.00071 0,
0.00038 -0.00092 0, 0 -0.001 0, -0.00038 -0.00092 0, -0.00071 -0.00071 0, -0.00092 -
0.00038 0, -0.001 0 0, -0.00092 0.00038 0, -0.00071 0.00071 0, -0.00038 0.00092 0, 0
0.001 0, 0.00038 0.00092 0, 0.00071 0.00071 0, 0.00092 0.00038 0, 0.001 0 0 ]
                  }
                }
                ]
              }
              Transform {
                rotation 0 1 0 0.785
```

```
                  translation 0.36 -0.4 0.36
                  children [
                  DEF triangle Shape {
                    appearance Appearance {
                      material USE MineColor
                    }
                    geometry IndexedFaceSet {
                      coordIndex [ 0 1 3 -1, 0 1 2 -1, 0 2 3 -1, 1 3 2 -1 ]
                      solid FALSE
                      coord Coordinate {
                        point [ 0 0 0.0925, 0 0.2775 0.122, -0.0925 0.2775 0, 0.0925
0.2775 0 ]
                      }
                    }
                  }
                  ]
                }
                Transform {
                  rotation 0 1 0 -0.785
                  translation -0.36 -0.4 0.36
                  children [
                  USE triangle
                  ]
                }
                Transform {
                  rotation 0 1 0 2.355
                  translation 0.36 -0.4 -0.36
                  children [
                  USE triangle
                  ]
                }
                Transform {
                  rotation 0 1 0 -2.355
                  translation -0.36 -0.4 -0.36
                  children [
                  USE triangle
                  ]
                }
                Transform {
                  translation 0.4 -0.08 0.4
                  children [
                  DEF ring Group {
                    children [
                    Transform {
                      rotation 1 0 0 1.57
                      children [
                      Shape {
                        appearance Appearance {
                          material DEF grey Material {
                            diffuseColor 0.5 0.5 0.5
                            specularColor 1 1 1
                          }
                        }
                        geometry Extrusion {
                          beginCap FALSE
                          convex FALSE
                          creaseAngle 1.57
                          crossSection [ 0 0, 0.007 0, 0.018 0.025, 0.032 0.035, 0.04
  0.036, 0.04 0.04, 0 0.04, 0 0 ]
                          endCap FALSE
                          spine [ 0.001 0 0, 0.00092 -0.00038 0, 0.00071 -0.00071 0,
  0.00038 -0.00092 0, 0 -0.001 0, -0.00038 -0.00092 0, -0.00071 -0.00071 0, -0.00092 -
  0.00038 0, -0.001 0 0, -0.00092 0.00038 0, -0.00071 0.00071 0, -0.00038 0.00092 0, 0
  0.001 0, 0.00038 0.00092 0, 0.00071 0.00071 0, 0.00092 0.00038 0, 0.001 0 0 ]
                        }
                      }
                      ]
                    }
                    Transform {
                      rotation 0 1 0 0.7535
                      translation 0 0.035 0
                      children [
                      Shape {
                        appearance Appearance {
                          material USE grey
                        }
                        geometry Extrusion {
                          beginCap FALSE
                          creaseAngle 1.57
                          crossSection [ 0.01 0, 0.0092 -0.0038, 0.0071 -0.0071, 0.0038
  -0.0092, 0 -0.01, -0.0038 -0.0092, -0.0071 -0.0071, -0.0092 -0.0038, -0.01 0, -0.0092
  0.0038, -0.0071 0.0071, -0.0038 0.0092, 0 0.01, 0.0038 0.0092, 0.0071 0.0071, 0.0092
  0.0038, 0.01 0 ]
                          endCap FALSE
                          spine [ 0.03 0 0, 0.0276 -0.0114 0, 0.0213 -0.0213 0, 0.0114
  -0.0276 0, 0 -0.03 0, -0.0114 -0.0276 0, -0.0213 -0.0213 0, -0.0276 -0.0114 0, -0.03 0
  0, -0.0276 0.0114 0, -0.0213 0.0213 0, -0.0114 0.0276 0, 0 0.03 0, 0.0114 0.0276 0,
  0.0213 0.0213 0, 0.0276 0.0114 0, 0.03 0 0 ]
                        }
                      }
                      ]
                    }
                    ]
                  }
                  ]
                }
                Transform {
                  rotation 0 1 0 1.57
                  translation -0.4 -0.08 0.4
                  children [
                  USE ring
                  ]
                }
                Transform {
                  rotation 0 1 0 1.57
                  translation 0.4 -0.08 -0.4
                  children [
                  USE ring
                  ]
                }
                Transform {
                  translation -0.4 -0.08 -0.4
                  children [
                  USE ring
                  ]
                }
                Transform {
                  translation 0 -0.11 0
                  children [
                  Shape {
                    appearance Appearance {
                      material Material {
```

```
          diffuseColor 0.5 0.5 0.5
          specularColor 0.2 0.2 0.2
        }
      }
      geometry Cylinder {
        height 0.22
        radius 0.15
      }
    }
    ]
  }
  Transform {
    rotation 0 0 1 0.935
    translation -0.461 -0.04 0
    children [
    DEF hole Shape {
      geometry Cylinder {
        height 0.01
        radius 0.04
      }
      appearance Appearance {
        material Material {
          diffuseColor 0 0 0
        }
      }
    }
    ]
  }
  Transform {
    rotation 0 0 1 -0.935
    translation 0.461 -0.04 0
    children [
    USE hole
    ]
  }
  Transform {
    rotation 1 0 0 -0.935
    translation 0 -0.04 -0.461
    children [
    USE hole
```

```
                  ]
                }
                Transform {
                  rotation 1 0 0 0.935
                  translation 0 -0.04 0.461
                  children [
                  USE hole
                  ]
                }
                Transform {
                  translation 0 -0.48 0
                  children [
                  Shape {
                    appearance Appearance {
                      material USE MineColor
                    }
                    geometry Cylinder {
                      height 0.01
                      radius 0.787
                    }
                  }
                  ]
                }
                ]
              }
              Group {
                children [
                Transform {
                  rotation 1 0 0 1.57
                  scale 1.5 1.5 1
                  children [
                  Shape {
                    appearance Appearance {
                      material USE MineColor
                    }
                    geometry Extrusion {
                      beginCap FALSE
                      creaseAngle 157
                      crossSection [ 0.1 0, 0.22 -0.01, 0.2275 -0.05 0.2675 -0.05, 0.49
0.4, 0.49 0.47, 0.53 0.47, 0.53 0.48, 0 0.48, 0 0.22, 0.0675 0.22, 0.1 0, ]
```

```
                    endCap FALSE
                    spine [ 0.001 0 0, 0.00092 -0.00038 0, 0.00071 -0.00071 0,
0.00038 -0.00092 0, 0 -0.001 0, -0.00038 -0.00092 0, -0.00071 -0.00071 0, -0.00092 -
0.00038 0, -0.001 0 0, -0.00092 0.00038 0, -0.00071 0.00071 0, -0.00038 0.00092 0, 0
0.001 0, 0.00038 0.00092 0, 0.00071 0.00071 0, 0.00092 0.00038 0, 0.001 0 0 ]
                  }
                }
                ]
              }
              Transform {
                rotation 0 1 0 0.785
                translation 0.36 -0.4 0.36
                children [
                USE triangle
                ]
              }
              Transform {
                rotation 0 1 0 -0.785
                translation -0.36 -0.4 0.36
                children [
                USE triangle
                ]
              }
              Transform {
                rotation 0 1 0 2.355
                translation 0.36 -0.4 -0.36
                children [
                USE triangle
                ]
              }
              Transform {
                rotation 0 1 0 -2.355
                translation -0.36 -0.4 -0.36
                children [
                USE triangle
                ]
              }
              Transform {
                translation 0.4 -0.08 0.4
                children [
                DEF sphere Group {
```

```
    children [
    Transform {
      rotation 1 0 0 1.57
      children [
      Shape {
        appearance Appearance {
          material USE grey
        }
        geometry Sphere {
          radius 0.05
        }
      }
      ]
    }
    ]
  }
  ]
}
Transform {
  rotation 0 1 0 1.57
  translation -0.4 -0.08 0.4
  children [
  USE sphere
  ]
}
Transform {
  rotation 0 1 0 1.57
  translation 0.4 -0.08 -0.4
  children [
  USE sphere
  ]
}
Transform {
  translation -0.4 -0.08 -0.4
  children [
  USE sphere
  ]
}
Transform {
  translation 0 -0.11 0
```

```
          children [
          Shape {
            appearance Appearance {
              material Material {
                diffuseColor 0.5 0.5 0.5
                specularColor 0.2 0.2 0.2
              }
            }
            geometry Cylinder {
              height 0.22
              radius 0.15
            }
          }
          ]
        }
        ]
      }
      Transform {
        translation 0 -0.24 0
        children [
        Shape {
          appearance Appearance {
            material USE MineColor
          }
          geometry Cylinder {
            height 0.48
            radius 0.49
          }
        }
        ]
      }
      WorldInfo {
        title "Null node"
      }
      ]
    }
    ]
  }
}
# Example scene starts here, in case this prototype is examined.
```

```
NavigationInfo {
  type [ "EXAMINE" "ANY" ]
}
Background {
       groundColor  [1 1 1]
       skyColor [1 1 1]
}
MantaMine {
  viewpointDescription  "Manta mine 10m"
}
Viewpoint {
  description "Manta mine 50m (LOD breakpoint)"
  position 0 0 50
}
Viewpoint {
  description "Manta mine 99m (LOD breakpoint 100m)"
  position 0 0 99
}
```

LIST OF REFERENCES

Refracted VRML Resource Center, "What is VRML" and "History of VRML," 2001. Available at http://www.refraction.com/vrml

Smith, James, "Floppy's VRML Guide," Vapour Technology Ltd., 2001. Available at http://www.vapourtech.com/vrmlguide

Web3D Consortium, "Extensible 3D (X3D™) Graphics Task Group," 1999-2001. Available at http://www.web3d.org/x3d.html

Ames, Andrea L., Nadeau, David R. and Moreland, John L., "The VRML 2.0 Sourcebook", Wiley Computer Publishing, New York, 1997. Available at http://www.wiley.com/legacy/compbooks/vrml2sbk/cover/cover.htm

Brutzman, Don, 2001, "Virtual Reality Modeling Language (VRML)", course home page. Naval Postgraduate School, Monterey California. Available at http://web.nps.navy.mil/~brutzman/vrml

Web3D Consortium, *The Virtual Reality Modeling Language, Annex C (normative), ECMAScript scripting reference*, 1997. Available at http://www.web3d.org/Specifications/VRML97/part1/javascript.html#Language

Brutzman, Don, "X3D-Edit for Extensible 3D (X3D) Graphics," 2001. Available at http://www.web3d.org/TaskGroups/x3d/translation/README.X3D-Edit.html

Marco, David B. and Healey, Anthony J., "Current Developments in Underwater Vehicle Control and Navigation: The NPS ARIES AUV," Naval Postgraduate School, Monterey California, 2001.

Brutzman, Donald P., "A Virtual World for an Autonomous Underwater Vehicle," Dissertation, Naval Postgraduate School, Monterey California, 1994.

Healey, A. J., Wu, J. and Brutzman, D. P., "Tactical Decision Aids Using Modeling and Simulation", Grant # N0001400WR20003, Naval Postgraduate School, Monterey California, 2001.