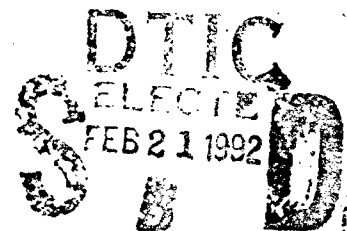
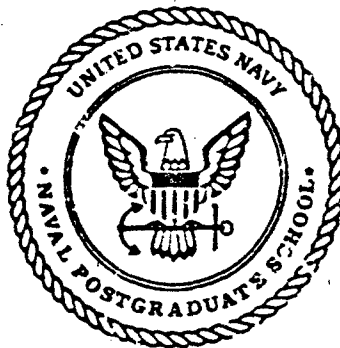


2

NAVAL POSTGRADUATE SCHOOL

Monterey, California

AD-A246 184



THESIS

A REAL TIME
AUTONOMOUS UNDERWATER VEHICLE
DYNAMIC SIMULATOR

by

Thomas A. Lewicz

December 1990

Thesis Advisor:

Michael J. Zyda

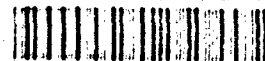
Approved for public release; distribution is unlimited.

Reproduced From
Best Available Copy

20000 831028

92 2 19 033

92-04339



REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			
4. PERFORMING ORGANIZATION REPORT NUMBER(S) .		5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b. OFFICE SYMBOL (if applicable) CS/ZK	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000		7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code)		10. SOURCE OF FUNDING NUMBERS	
		PROGRAM ELEMENT NO.	PROJECT NO.
		TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

A REAL TIME AUTONOMOUS UNDERATER VEHICLE DYNAMIC SIMULATOR (unclassified)

12. PERSONAL AUTHOR(S)

Jurewicz, Thomas A.

13a. TYPE OF REPORT

Master's Thesis

13b. TIME COVERED

FROM 04/89 TO 12/90

14. DATE OF REPORT (Year, Month, Day)

December 1990

15. PAGE COUNT

91

16. SUPPLEMENTARY NOTATION

The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

Graphics, Dynamics, Real Time, Simulation, Path Planning,
DoD Software Development

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The NPS Autonomous Underwater Vehicle Simulator is a joint project between the Naval Postgraduate School's Mechanical Engineering and Computer Science Departments. In order to test mission planning and execution software, an accurate vehicle dynamic model is required. Using dynamics based upon the Navy's Swimmer Delivery Vehicle (SDV), there is a need to continually update the hydrodynamic coefficients based upon actual vehicle- in - water testing. The NPS AUV Dynamic Simulator contains a full set of submarine equations of motion and hydrodynamic coefficients. The coefficients are modifiable on-line, and a replay capability exists for further performance review. Using Monterey Bay as an underwater testing environment, there is the need to be able to display expansive terrain data while maintaining the real time simulation. The Variable Terrain Resolution Algorithm incorporated into the NPS AUV Dynamic Simulator enables the entire Monterey Bay data base to be displayed in real time. Resolution adjustments are made automatically based upon the vehicle's depth level and system performance.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT

☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION

UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL

Michael J. Zyda

22b. TELEPHONE (Include Area Code)

(408) 646-2305

22c. OFFICE SYMBOL

52Zk

Approved for public release; distribution is unlimited

**A REAL TIME
AUTONOMOUS UNDERWATER VEHICLE
DYNAMIC SUMULATOR**

by

Thomas A. Jurewicz
Commander, United States Navy
B.S., Ocean Engineering
United States Naval Academy, 1975

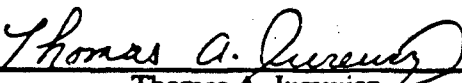
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF COMPUTER SCIENCE

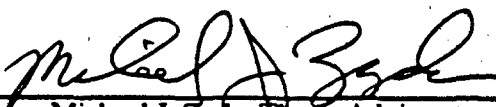
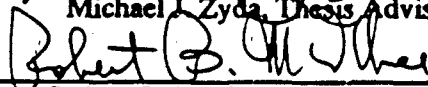
from the

NAVAL POSTGRADUATE SCHOOL
December 1990

Author:


Thomas A. Jurewicz

Approved By:


Michael J. Zyda, Thesis Advisor

Robert B. McGhee, Second Reader

Robert B. McGhee, Chairman,
Department of Computer Science

ABSTRACT

The NPS Autonomous Underwater Vehicle Simulator is a joint project between the Naval Postgraduate School's Mechanical Engineering and Computer Science Departments. In order to test mission planning and execution software, an accurate vehicle dynamic model is required. Using dynamics based upon the Navy's Swimmer Delivery Vehicle (SDV), there is a need to continually update the hydrodynamic coefficients based upon actual vehicle in-water testing. The NPS AUV Dynamic Simulator contains a full set of submarine equations of motion and hydrodynamic coefficients. The coefficients are modifiable on-line, and a replay capability exists for further performance review.

Using Monterey Bay as an underwater testing environment, there is the need to be able to display expansive terrain data while maintaining the real time simulation. The Variable Terrain Resolution Algorithm incorporated into the NPS AUV Dynamic Simulator enables the entire Monterey Bay data base to be displayed in real time. Resolution adjustments are automatically based upon the vehicle's depth level and system performance.



Accession For	
NTIS	☒
DTIC	☐
Unpublished	☐
Justification	
By	
Distribution	
Availability Codes	
Dist	Avail and/or Special
A-1	

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	BACKGROUND	1
B.	FOCUS -- NPS AUV DYNAMIC SIMULATOR	1
C.	THESIS ORGANIZATION	2
II.	SURVEY OF PREVIOUS WORK	4
A.	NPS SIMULATOR REVIEW	4
1.	Fiber Optically Guided Missile (FOGM) Simulator	4
2.	Vehicle (VEH) Simulator	4
3.	Moving Platform Simulator (MPS)	5
4.	Forward Observer Simulation Trainer (FOST)	5
5.	Command and Control Workstation of the Future (CCWF)	6
6.	CCWF, Subsurface and Periscope Views	6
7.	NPSNET	7
B.	AUV SIMULATOR DEVELOPMENT	8
1.	Use of SDV Hydrocoefficients	8
2.	Origins of NPS Auv Simulator	8
3.	NPS AUV-SIM1	8
4.	NPS AUV-SIM2	9
C.	NPS AUV-III	9
III.	GRAPHICS PIPELINE LOAD REDUCTION TECHNIQUES ..	11
A.	MESH DRAWING ROUTINES	11
B.	VARIABLE TERRAIN RESOLUTION	13
C.	POLYGON CULLING	13
IV.	VARIABLE TERRAIN RESOLUTION ALGORITHM	18
A.	BACKGROUND	18

B.	ADVANTAGES OF USING VTRA	19
1.	Compatible with DMA Terrain Database	19
2.	Less Storage Requirements	21
3.	VTRA improves DMA terrain rendering efficiency	21
4.	VTRA resolution can be adjusted automatically	24
5.	VTRA Adjusts to Workstation Upgrades	25
C.	ADDITIONAL VTRA DRAWING CONSIDERATIONS	25
1.	Seam Filling	25
2.	Geographic Referencing	26
3.	Inner Horizon Blocking Of Draw Routine	26
4.	Viewer Perspective	26
5.	Terrain Features	27
D.	VTRA BENCHMARKING	27
V.	AUV DATA STRUCTURE	32
A.	INTRODUCTION	32
B.	AUV_POLYGONS STRUCTURE	33
C.	DYNAMICS STRUCTURE	34
1.	Delta_time	34
2.	Forces	35
3.	Inverse Mass Matrix	35
4.	H-matrix	37
5.	T-matrix	38
D.	VEHICLE GEOMETRY STRUCTURE	39
1.	Mass Matrix	39
2.	Other AUV Geometry Considerations	40
E.	COEFFICIENTS STRUCTURE	40
F.	SURFACES STRUCTURE	42
VI.	AUV DYNAMICS	45
A.	INTRODUCTION	45
1.	Dynamics, Animation, and Simulation	45

a.	Restrict motions to those which are realistic.	45
b.	Portrays complex motion with minimal user input.	45
c.	Dynamic constraints can be automatically imposed.	45
d.	Move complex bodies in a natural way.	45
2.	How to Employ Dynamics	45
a.	Build dynamic equations of motion	46
b.	Solve equations for forces and accelerations.	46
c.	Determine velocities and positions through integration.	46
d.	Update the object's state.	46
B.	AUV EQUATIONS OF MOTION	46
1.	Viscous Crossflow Forces	46
2.	Equations Format	48
C.	SOLVING FOR FORCES, TORQUES, & ACCELERATIONS	55
D.	SOLVING FOR VELOCITIES & POSITION CHANGES	56
E.	UPDATING THE AUV'S STATE	58
1.	Create the incremental H matrix	58
a.	Rotation Order Matters	58
b.	Vehicle Coordinate System Alignment	59
2.	Revising the Homogeneous Transform Matrix	60
3.	Extracting Pitch, Roll, and Heading information	61
F.	DYNAMICS AND REAL TIME APPLICATIONS	63
1.	Dynamics is not the Limiting Factor	63
a.	Dynamic and Non-dynamic Modes	63
b.	Dynamic Mode Benchmarks	63
2.	Parallel Processing	64
3.	Addition of Dynamic Constraints	65
VII.	NPS AUV SIMULATOR	66
A.	USER INTERFACE	66
B.	MASTER SELECTION PANEL	67
C.	MOUSE PANEL	68

D. PERFORMANCE PANEL	69
E. FRAMES PANEL	70
F. RECORDER PANEL	71
G. VELOCITIES PANEL	72
H. BOTTOM CONTOUR PANEL	73
I. TERRAIN PANEL	74
VIII. FUTURE DIRECTIONS	76
A. DYNAMIC CONSTRAINTS AND PARALLEL PROCESSING	76
B. INCORPORATION OF PERIPHERAL PACKAGES	76
1. Controller	76
2. Navigator	77
3. Mission Planner/Replanner	77
C. TERRAIN	77
D. AUV MODEL DRAWING	78
E. CONCLUSIONS	78

ACKNOWLEDGEMENTS

The NPS AUV Dynamic Simulator would have not been possible without the close cooperation exhibited between the Computer Science and Mechanical Engineering Departments. Dave Marco provided education on submarine equations of motion, and their adaptation to the AUV. Fotis Pappoulas patiently reworked the equations and the hydrocoefficients for adaptation from the SDV to the AUV.

Dave Fratt provided insight into terrain rendering algorithms and their applicability in real time graphics programming. His knowledge of "C" programming language and the UNIX operating system helped bring the project to fruition.

Lieutenant Commander Rich Prevatt and Lieutenant Dave King provided invaluable assistance in helping incorporate their Panel Designer and Toolbox while still under development. Many program enhancements were made possible by the rapid prototyping capability of their project.

Dr. Robert McGhee, who sparked my enthusiasm in the AUV project, provided a solid foundation in rigid body dynamics and kinematics. His patience, vision, and expertise were invaluable during the past year.

Dr. Michael Zyda, my principal advisor, provided the insight into the value of real time graphics applications. His guidance provide direction throughout the project development.

Most importantly, my special thanks to my wife Lana, without whose sacrifices, love, and support, this would not have been possible.

I. INTRODUCTION

A. BACKGROUND

There is a growing effort within the Department of Defense to develop autonomous underwater vehicles. Without the need to incorporate life support systems, there is promise that an AUV can do a variety of missions at less expense, and without danger to human life. During software and hardware development, there is a risk of loss of an autonomous underwater vehicle if actually deployed, therefore, software must be thoroughly tested, preferably in its expected environment. With the advent of high speed, low cost graphics workstations, it is now possible simulate submarine dynamics in real time. Controller and mission planning software can be tested and real time feedback obtained. Using underwater grid terrain data, such as that available from the Defense Mapping Agency, missions can potentially be simulated anywhere in the world, at any depth. Various mission factors such as changing currents, unplanned obstacles, and vehicle control surface failure can be observed prior to executing various missions. By incorporating the AUV into simulators such as the NPSNET (Zyda, Pratt 1990), vehicle missions can be executed in conjunction with a coordinated operations scenario. In order to run these missions in real time, algorithms need to be developed to make optimum usage of the workstation's graphics capabilities.

B. FOCUS -- NPS AUV DYNAMIC SIMULATOR

This thesis concentrates on two main aspects. The first is to develop the ability to generate accurate hydrodynamic coefficients and submarine equations of motions. The second is to portray the vehicle in its anticipated environment in real time.

By accurately predicting vehicle performance, system software can be developed and tested prior to incorporation into the actual vehicle. The NPS AUV Dynamic Simulator enables the user to record vehicle performance with any set of hydrocoefficients. System response on the simulator can be compared to in-water vehicle testing, coefficients adjusted on-line, and simulator performance observed until it emulates the actual vehicle. Through this bootstrapping effect, an accurate graphics model can be developed without the need to perform expensive test-tank operations.

The Monterey Bay database was used to develop the terrain rendering algorithm utilized in the NPS AUV Simulator. The proximity of the bay to the Naval Postgraduate School (NPS) combined with the interesting subterrain features of the Monterey Bay Canyon, make the bay a logical choice for future test runs of the actual vehicle. Mission planning systems can be used to generate proposed paths through the canyon. Bay currents can be incorporated into the model. While most of the actual vehicle testing is done in the constraints of the NPS swimming pool, full dynamic and artificial intelligence software testing require a more expansive area. The Variable Terrain Resolution Algorithm developed herein can be ported to other simulators using DMA Digital Terrain Data such as NPS Command and Control Workstation of the Future (Weeks, Phillips 1989).

C. THESIS ORGANIZATION

Chapter II provides a background on other vehicle simulators developed at NPS. The development and refinement of terrain rendering algorithms is traced through the various simulators at NPS. The use of dynamics to graphically model the NPS AUV is traced from the simulator's origin to the current model NPS AUV III.

Chapter III describes the techniques utilized in the NPS AUV Simulator to portray the environment in real time. These techniques include high speed terrain drawing routines, variable terrain resolution display, and field of view culling.

Chapter IV provides details and performance measurements of the Variable Terrain Resolution Algorithm (VTRA) used to display Monterey Bay. VTRA is a recursive, binary reduction technique for displaying grid terrain about an observer to the horizon.

Chapter V describes the AUV data structure. By taking an object oriented approach, the submarine can inherit rigid body properties while maintain those unique to a submarine environment.

Chapter VI discussed the dynamics model used for AUV III. The AUV equations of motion and hydrodynamic coefficients are described. The procedure for converting thrusters rpm and fin deflections to vehicle motion is discussed.

Chapter VII details how to operate the AUV Simulator. The User Interface is described along with the system capabilities.

Chapter VIII provides the limitations and future direction of the project.

II. SURVEY OF PREVIOUS WORK

A. NPS SIMULATOR REVIEW

1. Fiber Optically Guided Missile (FOGM) Simulator

The FOGM simulator, implemented on the SGI IRIS 3120 workstation, featured a 10 kilometer by 10 kilometer grid terrain data base of Fort Hunter Liggett, California (Smith 1987). The data was presented as 3-sided polygons using a Painter's algorithm where each polygon is scan converted drawing the closest polygons last while "painting over" further polygons. All terrain was drawn at a constant resolution, including terrain hidden from the observer. FOGM maintained a frame rate of three to four frames per second by displaying every 6th data point. Elevation data was exaggerated in order to provide a more hilly appearance, with terrain coloring made a function of elevation. Flat shading was implemented instead of Gouraud shading in order to provide a "checkerboard effect" aiding motion detection. FOGM implemented trivial culling by establishing a rectangular bounding volume based upon the vehicle's heading. Polygons outside this volume were culled.

2. Vehicle (VEH) Simulator

VEH, an extension of the FOGM simulator, was also implemented on an IRIS 3120 and completed in December 1987. The FOGM terrain rendering algorithm was modified to include a more refined Field of View (FOV) culling algorithm. Full 3D culling often uses clipping planes to form a four sided "pyramid of vision" with the axis of the pyramid along the vehicle's path. VEH utilized the left, right, and far clipping planes. VEH subjected FOGM's bounding volume to further culling based upon a narrow or wide field of view. Only those polygons within the bounding volume, and within the FOV were sent through the graphics pipeline. Vehicle roll (viewer "twist") was not incorporated in either

FOGM or VEH thus eliminating the need for more than three clipping planes. FOGM's FOV culling was strictly a function of rotation about the screen's "y" axis (vehicle heading). A further contribution of VEH was the incorporation of SINE, COSINE, and TANGENT lookup tables instead of function calls. Speed improvements of over 1000% were noticed in some cases when using the tables instead of the math library (Oliver 1988).

3. Moving Platform Simulator (MPS)

The MPS series evolved from the VEH and FOGM simulators. Significant improvements were made to the gridded terrain algorithms previously used. Implemented on an IRIS 4D/70GT workstation with hardware supporting the z-buffer algorithm, the need for the Painter's algorithm for hidden surface elimination was removed.

MPS incorporated three resolution levels, and variable field of views. The highest resolution level extends out to a distance which is a function of the field of view. The resolution is then halved and extended out 2000 meters, halved again out to the horizon. Gaps occur at the seams between resolution levels as extra vertices exist on the high resolution side of the seam. To eliminate this problem, extra polygons were drawn to fill the holes. These polygons, termed "skirts", were vertical planes which were often perpendicular to the actual terrain. A drawback to this method was that as the seams changed location with vehicle movement, a slight flickering would occur due to the insertion and deletion of these vertical surfaces.

Whereas previous simulators were limited to the 10 by 10 kilometer area of Fort Hunter Liggett, MPS had the ability to display gridded terrain databases containing any grid point spacing.

4. Forward Observer Simulation Trainer (FOST)

FOST (Drummond 1989) utilized DMA Level 1 Digital Terrain Elevation Data to generate coordinates in the Military Grid Reference System (MGRS). Although FOST

adapted the MPS terrain rendering algorithm, minor modifications included switching from polygon primitives to *mesh* primitives to increase performance. This decision was based upon performance measurements in MPSII (Winn June 89), the second simulator in the MPS series. Paragraph 6 describes mesh drawing in more detail.

5. Command and Control Workstation of the Future (CCWF)

The CCWF (Harris 1988) was initiated at the Naval Postgraduate School as part of an effort to provide a 3D real time interface for the Battle Group Commander. In order to enhance real time capabilities, CCWF also incorporated variable terrain resolution strategy. While MPS adapted binary reduction between resolution levels, CCWF decreased resolution levels from 100 yard spacing to 1200 yard spacing and finally 12000 yard separation at the lowest resolution level. Such a strategy provided a greater reduction of polygons at lower resolutions but provided a more dynamic terrain change at the seams. In order to maintain three separate resolution levels of data, separate terrain databases were created for the various resolutions. While increasing data storage requirements, the reduction of run time calculations resulted in an increased frame rate.

To solve the boundary problem, CCWF utilized the "skirt" method developed in the MPS series to draw seams between resolution levels. All three resolution levels are drawn from the vehicle, resulting in the high resolution carpet being drawn over lower resolution carpets. The net effect was that lower resolution terrain would "cut through" valleys of the higher resolution terrain.

The CCWF was the first NPS Simulator to draw data using DMA's Digital Terrain Elevation Data. The area of operations centered around the Sea of Japan.

6. CCWF, Subsurface and Periscope Views

In follow-up work to the CCWF (Weeks 1989), a triangular mesh drawing routine was incorporated in addition to the polygon drawing routine. By reducing the number of

vertices required to be sent through the graphics pipeline (4 instead of 6 per triangle pair), a 50% speed increase in graphics frame rate was obtained. When using mesh drawing routines, vertex normals instead of polygon normals were generated resulting in a smoother, more realistic appearance. The "skirt" method of filling resolution seams did not work as well when using the "mesh" mode. Since the skirt lighting normals were horizontal, skirt flickering discovered in MPS became very pronounced as surrounding lighting normals were essentially vertical. The polygon (checkerboard) method was left available as an option. Without terrain features, motion on level surfaces is often difficult to detect without the checkerboard effect.

A primary concern is the storage requirement for CCWF lighting. To adequately light the terrain, vertex normals are computed at start-up. Thus terrain database requirements grew from 2.88 megabytes to well over 21 megabytes. An attempt was made to compute normals dynamically, however, a 50% performance reduction degraded the system real time performance capabilities.

A recommendation for future research was to incorporate control surfaces into the ships, such as rudders. Initial work on the AUV NPS Simulator was undertaken with CCWF requirements well understood; specifically applicable to CCWF are (1) use of control surfaces to drive vehicle; (2) elimination of requirement to use skirts at resolution boundaries; (3) incorporation of sonar; (4) development of vehicle control panel.

7. NPSNET

NPSNET (Zyda, Pratt 1990) is the Naval Postgraduate School's low-cost version of the DARPA SIMNET System. While refining many of the NPS terrain rendering algorithms, NPSNET's enhancements include incorporation of cultural terrain features such as ground cover, man-made structures, and terrain texturing. Research continues on

optimal display of such features on sloped and variable resolution terrain while maintaining real-time updates.

B. AUV SIMULATOR DEVELOPMENT

1. Use of SDV Hydrocoefficients

The original NPS non-graphical simulation of an underwater vehicle can be found in (Boncal, 1987). Utilizing basic submarine equations of motion (Gertler 1967), with modifications to reflect the geometry of the U.S. Navy's Swimmer's Delivery Vehicle (Smith 1978), Boncal designed a controller to control rudders, bow planes, and stern planes based upon the vehicle's predicted dynamics.

2. Origins of NPS Auv Simulator

The initial 3D graphics submarine simulator (MacPherson 1988) consisted of a submersible which utilized a rudder, stern planes, and a single screw. Movement of these control surfaces imparted pitch, yaw, and speed to the vehicle. Simple dynamics were employed to derive appropriate vehicle responses. Actual submarine dynamics were not modeled as the simulator was utilized to test mission path planning algorithm's only.

3. NPS AUV-SIM1

The current AUV graphics simulator is an extension of a graduate graphics project by D. Marco, R. Rogers, and M. Schwartz (Zyda, McGhee, Kwak 1990). AUV-SIM1 was the first graphics simulator to utilize the Swimmer's Delivery Vehicle equations of motion as modified by R. Boncal (Boncal 1987). Intended to model the NPS Autonomous Underwater Vehicle, the simulator demonstrated realistic submarine dynamic behavior. The simulator incorporated a mouse panel to make adjustments to rpm, rudder, and bow planes. The environment was a 120 ft x 60ft x 8ft swimming pool, the vehicle drawn as a six foot submersible with twin screws, stern rudders, bow and stern planes.

Communications code was added to receive autopilot command controls from a Symbolics LISP machine (Nordman 1989).

4. NPS AUV-SIM2

The appearance of the graphics AUV was modified to reflect the actual AUV being built by the Naval Postgraduate School. Essentially symmetric in shape, AUV-SIM2 has eight control surfaces consisting of bow planes, stern planes, bow rudders, and stern rudders. The swimming pool was redesigned to reflect the appearance of the NPS swimming pool where initial testing of the actual vehicle would take place. The basic "C" code was further modularized. The Mission Planning Expert System was developed on the Symbolics Lisp Machine using the KEE Expert Shell resulting in some modifications to the "C" communications code (Ong 1989). Although the vehicle's appearance reflected the actual AUV, the dynamics and geometry reflected the asymmetrical, much larger SDV.

C. NPS AUV-III

NPS AUV-III is a result of this thesis. While making minor upgrades to the vehicle's appearance, the primary contributions were a reengineering of the software to encapsulate the AUV as a rigid body using object oriented programming techniques prototyped in LISP. The equations of motions and hydrodynamic coefficients were modified to reflect the geometry of the NPS AUV rather than the SDV. Furthermore, the drag coefficients and added-mass coefficients are no longer "hard coded" into the program, but are parsed from an external file at the program's initiation. These coefficients are modifiable on-line so adjustments can easily be made and tested. The revised coefficients can then be saved to an external file for further refinement. The Monterey Bay environment was incorporated to allow more expansive testing of the search algorithms and testing of system dynamics that can not be tested within the constraints of the NPS Pool.

A record capability exists to develop a script that can be used by the actual vehicle during testing. A replay capability exists to reexamine missions, or to test externally generated scripts. A sliding scale enables the speed of replay to be adjusted. On line stripcharts display changes in velocities and accelerations to be displayed along all axes. The user interface was designed to allow to display the vehicle's orientation (pitch, heading, and roll).

III. GRAPHICS PIPELINE LOAD REDUCTION TECHNIQUES

The Silicon Graphics 4D/240GTX contains four MIPS R3000 CPU's and R3010 RISC components. The CPUs run at 25mhz and together execute approximately 80 million instructions per second (MIPS) achieving four double precision Mflops (Ackley 1989). The graphics architecture is divided into four subsystems: the transformation subsystem, scan-conversion subsystem, raster subsystem, and display subsystem. Of interest here is the transformation subsystem, for this is where the limit is set on the number of vertices which can be generated per second.

The transformation subsystem, called the Geometry Engine, is capable of processing 400,000 vertices per second. A single vertex transformation requires approximately 100 FLOPS. To achieve a frame rate of 10HZ, for example, we must attempt to pass less than 40,000 vertices per frame. 30 % of the Geometry Engine's work is in performing vertex transformations, with the remaining work performing operations such as lighting calculations and normal transformations. Since the programmer can directly influence the total number of vertices sent to the Geometry Engine, it is often desirable to employ vertex reduction techniques when the goal is real time graphics display. Some of the techniques used in the NPS AUV simulator are described below.

A. MESH DRAWING ROUTINES

In order to draw the entire Monterey Bay database as polygons (triangles), each internal vertex needs to be sent six times, once for each polygon that shares the vertex. As demonstrated in CCWF, the graphics library function *bgtmesh()* can greatly improve graphics pipeline efficiency. By drawing the area as a series of mesh strips, each internal vertex needs to be sent only twice to represent the six adjacent polygons, as illustrated in Figure 3.1. The vertex information is maintained in two "vertex registers" within the IRIS-

4D. As a result, the total number of vertices required drops from over 300,000 to slightly over 100,000. Since the Geometry Engine is capable of 400k vertices per second, it can pass 135k triangles per second when using mesh drawing routines.

The IRIS-4D VGX models contain three vertex registers, enabling the vertices to be represented as part of a quadrilateral using *bgnqstrip()*. The function *bgnqstrip()* increases the efficiency of the Geometry Engine even further, as 100k quadrilaterals (200k triangles) can be drawn per second. In addition, "Q-mesh" provides superior shading and lighting over "T-mesh" (Graphics Library 1990).

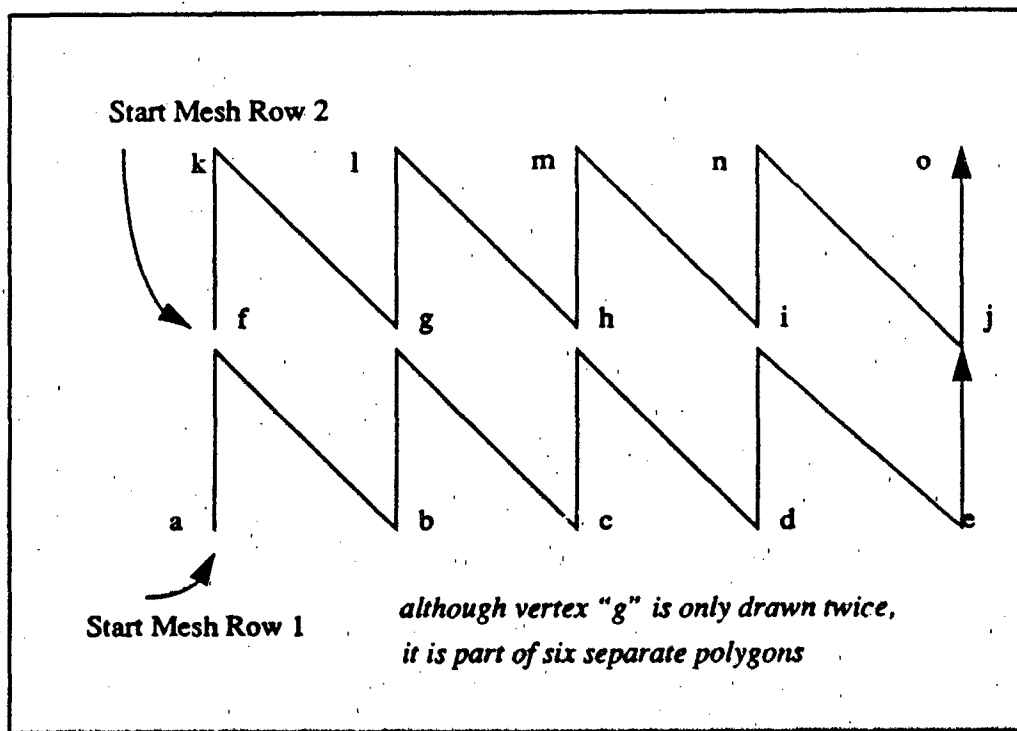


Figure 3.1 Mesh Drawing Advantage

B. VARIABLE TERRAIN RESOLUTION

Both CCWF and MPS demonstrated the importance of variable terrain resolution. Since the number of data points available for display increase by with the square of the distance from the observer, it is essential that such a reduction be incorporated. By limiting the degree of the resolution changes and increasing the number of resolution changes, the NPS AUV Dynamic Simulator is able to reduce the rate of vertex increase from $O(N^2)$ to nearly $O(N)$, with the later being approached as the number of resolution levels is increased. A major concern with multiple resolutions is "seam handling", or the smooth transition from one resolution to another. The chapter on the VTRA (Variable Terrain Resolution Algorithm) addresses this issue. Figure 3.2 depicts the effect of decreasing the resolution in a binary fashion, while increasing the number of resolution levels between the observer and the horizon.

C. POLYGON CULLING

The VTRA can be applied in conjunction with culling. The merits of culling have been demonstrated in numerous NPS simulators. As an example of the power of culling, consider a 60 degree Field of View. If polygons outside the FOV can be culled, then a lessened load is placed on the graphics pipeline. Very efficient mesh drawing routines now exist, so one must weigh the CPU overhead required of culling, against the efficiency of the Geometry Engine. Often, a "rough clip" is superior to fine culling since the later must often be done in the mesh drawing loop, with culling conditions checked against every vertex. Fine culling was experimented with in the NPS AUV Simulator, and performance was actually lower than that with the "quadmesh" drawing without culling.

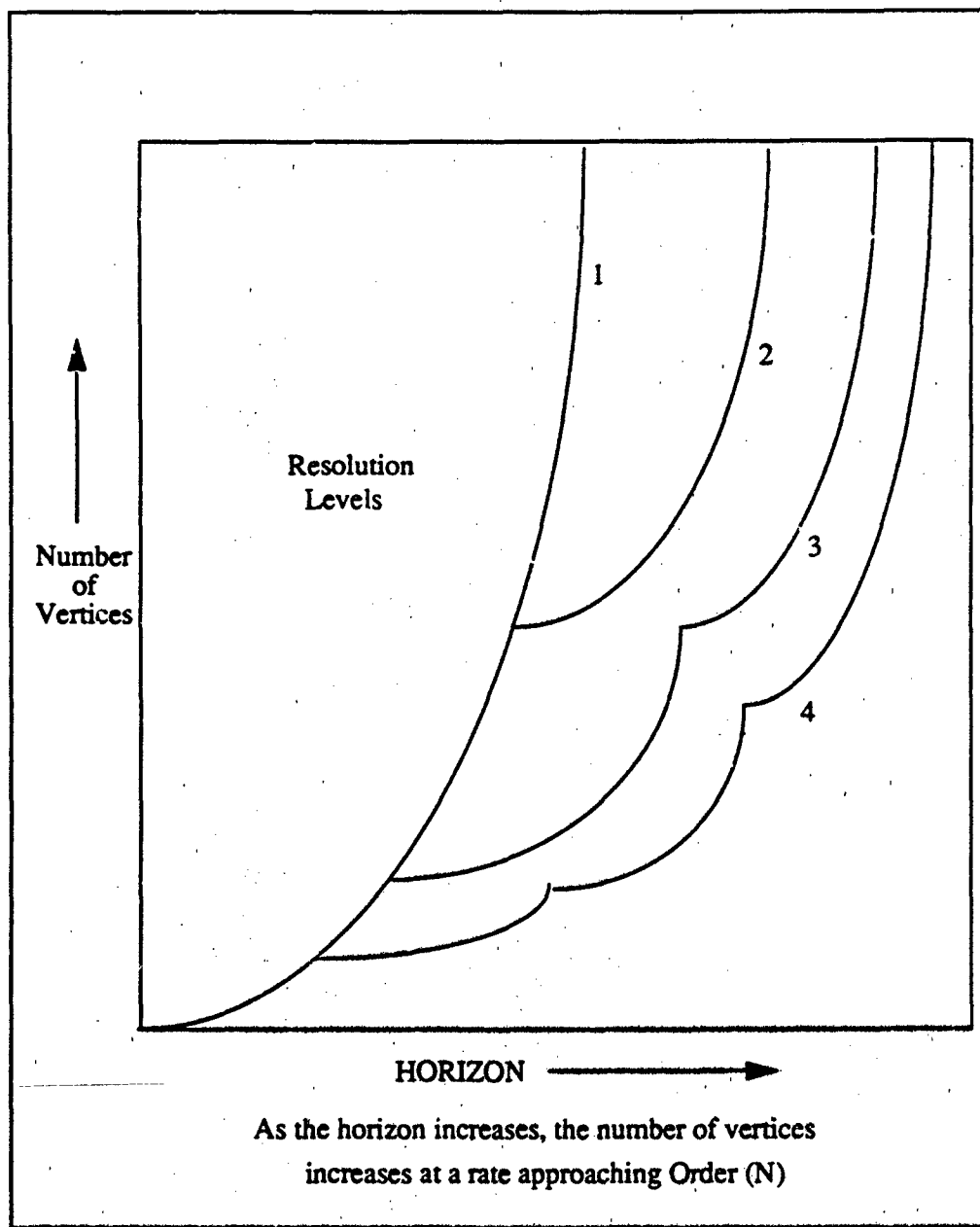


Figure 3.2 Variable Resolution Effects on Vertex Count

Figure 3.3 is an aerial viewer of the Monterey Bay. The Submarine heading is 045 degrees with a field of view of ± 45 degrees. Data outside the FOV is culled

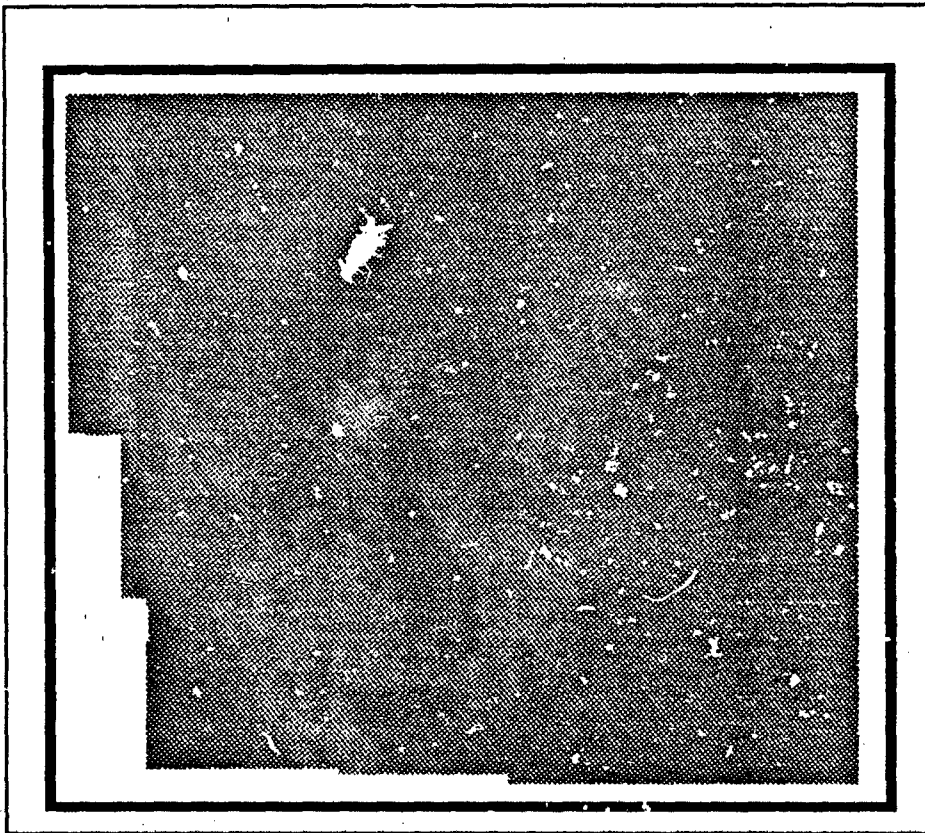


Figure 3.3 Monterey Bay Culled

The culling in the NPS Simulator is similar to the CCWF culling algorithm. The terrain is divided into four sectors, North, South, East, and West. The origin is at the viewer's location (in NPS AUV Simulator, the observer need not be at the vehicle). The view direction is the vehicle's heading, or the observer's viewing azimuth. Sine and Cosine lookup tables are created the first time the culling routine is activated. VTRA has the notion

of minimum and maximum rows and columns, usually the limit of the database. The NPS AUV Simulator simply uses the viewing azimuth to further constrain these limits. For example, if the observer is viewing towards the East sector, then the minimum column can immediately be increased to the observer's column. The maximum column is already set by the horizon limitations. Therefore, the only requirement is to adjust the maximum and minimum rows using the view direction, right and left clipping angles, horizon, and trigonometric lookup tables. Figure 3.4 contains the pseudocode for culling techniques utilized in the NPS AUV Simulator.

Since the NPS AUV Simulator is essentially a flight simulator and capable of angular accelerations along all three axes, further culling was attempted based upon sine of the pitch in conjunction with the altitude and the sine of the roll in conjunction with the vertical field of view. Although this was relatively easy to accomplish, the savings in vertex generation could not match the additional mathematics involved and system performance declined. Therefore, culling is based upon rotation around the system's "y" axis only (heading). With culling activated, the NPS AUV Simulator's frame rate increased by 2 to 4 frames per second depending on number of resolution levels and horizon.

```

if (NORTH_SECTOR) {
    minrow = viewer_row;
    maxrow = maximum_row;
    mincol = viewer_col - horizon * tan(left_clipping_azimuth);
    maxcol = viewer_col + horizon * tan(right_clipping_azimuth);
}
if (SOUTH_SECTOR) {
    minrow = minimum_row;
    maxrow = viewer_row;
    mincol = viewer_col - horizon * tan(right_clipping_azimuth);
    maxcol = viewer_col - horizon * tan(left_clipping_azimuth);
}
if (EAST_SECTOR) {
    minrow = viewer_row + horizon / tan(right_clipping_azimuth);
    maxrow = viewer_row + horizon / tan(left_clipping_azimuth);
    mincol = viewer_col;
    maxcol = maximum_col;
}
if (WEST_SECTOR) {
    minrow = viewer_row - horizon / tan(left_clipping_azimuth);
    maxrow = viewer_row - horizon / tan(right_clipping_azimuth);
    mincol = minimum_col;
    maxcol = viewer_col;
}

```

Figure 3.4 Clipping Pseudocode

The row and column constraints are reset by the clipping routine, and are utilized by VTRA for generating terrain.

IV. VARIABLE TERRAIN RESOLUTION ALGORITHM

A. BACKGROUND

As shown in the previous chapter, there is a need to display grid terrain at various resolutions if real time simulation is the goal. The variable terrain resolution algorithm (VTRA) was initially conceived while conducting research on aerial view techniques for the Command and Control Workstation of the Future, and is fully incorporated into the NPS AUV Simulator. The algorithm assumes that highest visibility terrain should be drawn around the observer, and that the observer's position be selectable, whether inside or outside a vehicle. The formula requires four inputs: the observer's horizon, the number of resolution levels required, the maximum resolution level required, and system performance as measured by "delta time" (the inverse of the system frame rate).

Based upon the input parameters, VTRA determines a grid density for the lowest resolution terrain, and draws this terrain from the horizon to $1/2$ horizon. The horizon is then reset to the $1/2$ horizon, and the grid density doubled. The function is called recursively with the new parameters until maximum density is achieved. This density is then drawn to the observer and the algorithm stopping condition achieved. Figure 4.1 contains the pseudocode for the algorithm as used in the NPS AUV Simulator.

```
/* vehicle is a structure containing the vehicle's state, i.e., orientation, position */
```

```
show_terrain(vehicle)
```

```
Veh_ptr vehicle;
```

```
{
```

```
    int start[0] = vehicle's X position on grid;
```

```
    int start[1] = vehicle's Y position on grid;
```

```
    int horizon = 128;
```

```
    int vert_spacing = 16;
```

```
    int max_res_level = 1;
```

```
    show2_terrain(vehicle, start, horizon, res_level);
```

```
}
```

```
show2_terrain(Veh_ptr vehicle, int start[2], int horizon, int vert_spacing)
```

```
{
```

```
    if(vert_spacing == max_res_level) {
```

```
        draw_terrain_from_horizon_to_vehicle_at_current_res_level();
```

```
    else {
```

```
        draw_terrain_from_horizon_to_1/2_horizon_at_current_res_level();
```

```
    }
```

Figure 4.1 VTRA Pseudocode

B. ADVANTAGES OF USING VTRA

1. Compatible with DMA Terrain Database

VTRA will work with any size two dimensional array of grid data. The value of using authentic DMA terrain data has been demonstrated in numerous NPG simulators. The algorithm was developed using Monterey Bay terrain data received courtesy of the Monterey Bay Research Institute (MBARI). Figure 4.2 depicts the data structure which was originally in values of positive meters and converted to negative feet. When the NPS AUV Simulator is activated, the function *scan_z_bay()* reads the data into a 222 x 245 x 3 array. The X and Y data is generated based on vertex spacing. Since above ground terrain is represented with zero elevation data, a random number generator assigns positive values to present a discernible coastline. Currently, work is underway to merge Monterey Bay subterrain data with DMA terrain data for use with VTRA and subsequent aerial and subsurface views of the Monterey Bay Coast.

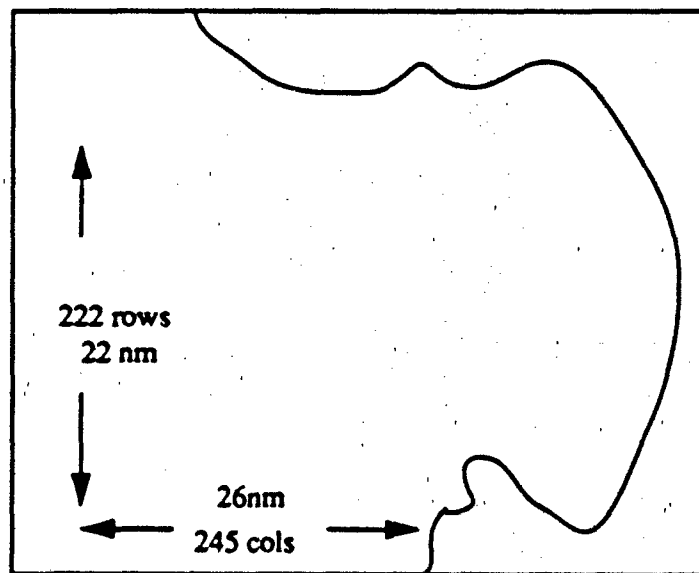


Figure 4.2 VTRA Pseudocode

2. Less Storage Requirements

VTRA uses an array of vertex normals generated during program initialization to perform lighting calculations. After the terrain data array is parsed, the NPS AUV Simulator uses the function *compute_bay_normals()* to generate vertex normals. Using the four adjacent vertices, normals for the four adjoining polygons are computed. From these four normals, a final unit vertex normal is generated. Unlike previous simulators using variable resolution strategies, there is no need to precompute and prestore various resolution data, only one data set need to be stored.

3. VTRA improves DMA terrain rendering efficiency

CCWF research (Weeks 1989) identified the amount of data required to display terrain out to the horizon, typically 26 nautical miles, as a major concern. Using DMA 1201 x 1201 terrain with 100 yard spacing as an example, we apply the VTRA to display the data base out to 26 nautical miles without using 100% of the database.

Placing the observer in the center of the grid, assume a horizon of 512 data points (51,200 yards or 25.5 nm), we apply the algorithm before field of view culling. Displaying all the data at highest resolution requires (1024 x 1024) or 1.2m vertices. Displaying all the data at 1/4 resolution requires (256 x 256) or 64k vertices. With the VTRA formula, and using 5 resolution levels, only 16k vertices are required while providing maximum resolution out to 3200 yards. This represents less than 2.0% of the total available vertices. By applying 4 resolution levels, the area of highest resolution is extended out to 6400 yards with a vertex count of 26k, less than 3% of the total available vertices.

When using DMA terrain data, one concern is when should another geographic cell be read from disk into memory. By extending the horizon, at low resolutions, a distance that is a function of the vehicle speed and the amount of time to recover data from storage, the horizon boundary can act as a "trigger" to initiate reading of a particular cell.

The flexibility of VTRA enables the programmer to emphasize either resolution or horizon. A geologist scanning a desert from the air would expect to see geological structures in greater detail as he approached the desert floor. As he descends, his horizon would decrease with the square root of the altitude.

There are three factors affecting the total vertex count using VTRA: horizon, total number of resolution levels, and maximum resolution level. Each of these factors is a VTRA input parameter.

The horizon is the number of data points extending from the observer that will be displayed on the screen. Data within the horizon is depicted after applying proper spacing. The horizon must always be a power of 2, i.e., 64, 128, 256, etc.. The data spacing factor must also be passed as a parameter so the algorithm can convert the horizon to geographical coordinates. Decreasing the horizon increases system performance.

The total number of resolution levels is only limited by available horizon. The lowest resolution extends from $1/2$ horizon to horizon, the next lowest from $1/4$ horizon to $1/2$ horizon. The binary reduction continues recursively until the maximum resolution level is reached. Data from the observer to the innermost horizon is depicted at maximum resolution level. Increasing the total number of resolution levels increases workstation performance, and decreases the total area of maximum resolution.

The maximum resolution level determines the density that terrain will be rendered closest to the observer. For example, at resolution level one, every data point is represented; at resolution level two, every other data point; and at resolution level four, every fourth point. Notice that there is no resolution level three, as VTRA relies upon a *binary reduction* of terrain resolutions. Each outer resolution area is $1/2$ the density of the inner resolution area, with the *innermost* resolution area representing the starting density. Therefore,

lowering the maximum resolution level from one, to two or four, will greatly reduce total vertex requirements.

The default parameters of the NPS AUV Simulator are a horizon of 256 data points, total number of resolution levels of four, and a maximum resolution level of one.

Figure 4.3 depicts a view of Monterey Bay from an altitude of 80 nautical miles. Figure 4.4 shows the bay drawn as a wiremesh with four VTRA resolutions .

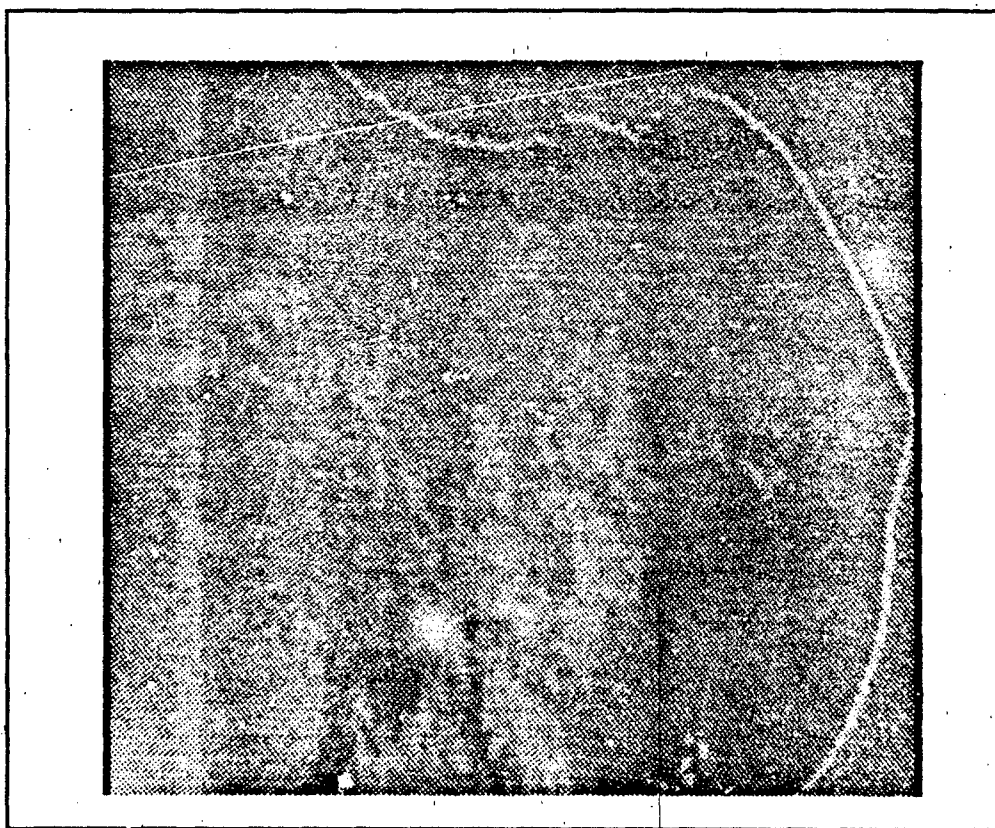


Figure 4.3 Filled View of Monterey Bay

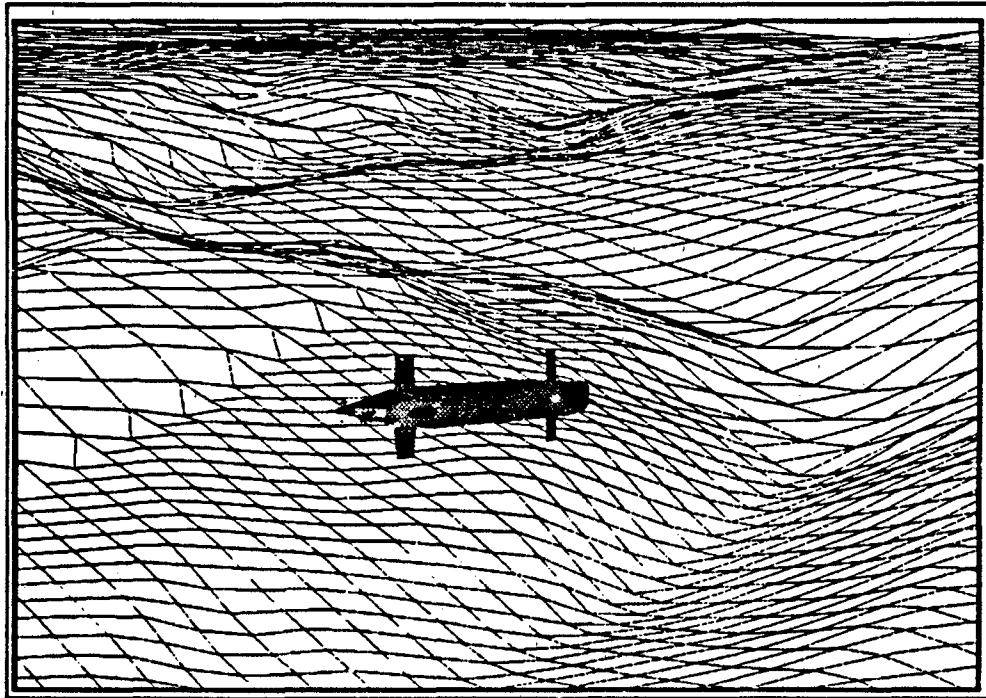


Figure 4.4 Four Resolutions of Monterey

4. VTRA resolution can be adjusted automatically

Since the three controlling factors are input parameters, the programmer can elect to create a function to automatically control these values. NPS AUV Simulator has such a function, *auto_resolution()*, which is activated from the *terrain control panel*.

The auv horizon is a function of its "absolute altitude", i.e., height above the terrain. As the vehicle climbs, the horizon doubles at programmed intervals. As the vehicle approaches "mountainous terrain", such as the Monterey Bay Canyon wall, the horizon will decrease by half to provide better resolution of the terrain.

The area of maximum resolution beneath the vehicle is inversely proportional to the vehicle horizon, i.e., as extra vertices are portrayed by extending the horizon, a

reduction of vertices directly below the vehicle takes place. The trade-off is nearly equal so system performance remains almost the same.

The third factor, number of resolution levels, is adjusted as a function of system performance. The instant a decrease of performance is detected, whether the cause is internal or external to the program, the number of resolution levels is increased easing the graphics pipeline load. Conversely, if the system is running efficiently, the number of resolutions is decreased, thus extending the range of higher density terrain rendering. The number of resolution levels becomes a function of workstation performance.

5. VTRA Adjusts to Workstation Upgrades

By adjusting the input parameters as a function of system performance as was done in the NPS AUV Simulator, discussed above, higher performance architectures using VTRA will maintain the real time depiction, and terrain rendering will automatically improve. There is no need to rewrite the program since VTRA can automatically provide more realistic terrain rendering.

C. ADDITIONAL VTRA DRAWING CONSIDERATIONS

1. Seam Filling

As seen throughout the development of vehicle simulators incorporating a variable resolution scheme, making a smooth transition from one resolution to another has been a problem with various solutions. Additional polygon generation and normal calculations required to "fill the seams" can degrade system performance. VTRA solved this problem by staying within the binary reduction recursive routine. NPS AUV Simulator seam "stitching" functions will work at any horizon. Rather than filling holes after the terrain has been generated, the seams are part of the terrain rendering process. Mesh drawing routines always require an even number of vertices from within the mesh function.

VTRA's stitching requires two vertices per seam for each vertex row scanned, ensuring this requirement is met. Every stitched vertex uses its precomputed normal for generating proper lighting effects. As a result, seams can not be detected as seen in the various terrain pictures throughout this paper.

2. Geographic Referencing

When drawing terrain at various resolutions, the choice of which points to represent should be a function both relative to the observer and relative to actual geographic position. In the original NPS AUV design, the points displayed were relative *only* to the observer. As a result, resolution level 4, for example, would "shift" the vertices displayed resulting in a rippling movement as the vehicle progressed. Steady terrain was generated by depicting only those vertices whose row % four and col % four equaled zero. Geographic referencing is also required in the seam drawing algorithm to ensure smooth blending from one resolution to another.

3. Inner Horizon Blocking Of Draw Routine

Previous simulators discovered problems when trying to place a small "high resolution carpet" over a lower resolution such as valleys being "sliced" by the underlying lower resolution plane. To ensure this doesn't happen, and to greatly reduce vertex generation, drawing of all data from horizon/2 to the observer is blocked, for each horizon in the recursive call except for the highest resolution which continues all the way to the observer.

4. Viewer Perspective

Resolution should be based on the observer position, not a vehicle's position. Therefore, the program should track the observer's coordinates, as these are the coordinates

the VTRA will base the recursive stopping condition, i.e., the center of high resolution display.

5. TERRAIN FEATURES

Drawing with mesh routines and lighting has generated realistic looking terrain in recent simulators. However, without the checkerboard effect, motion is difficult to sense. Terrain structures or vegetation can help provide this effect. This is evident in NPSNET which uses ground cover and structures to aid with the sense of motion.

While displaying features on highest resolution terrain does not cause a problem, at lower resolution the item may rest on a vertex not displayed, causing polygons to slice through the structure. A possible solution in VTRA is to modify the algorithm to display highest resolution terrain "patch" where any structure exists. The row scanning process can make this an easily incorporated feature.

D. VTRA BENCHMARKING

These figures were taken from the NPS AUV Simulator with the Simulator's "culling" feature disabled. The vehicle was operated from a "Cockpit View" so that the vehicle's polygons were not drawn. The Simulator was run on a Silicon Graphics, Inc., IRIS 4D/240VGX Workstation rated 80 MIPS and 16 M FLOPS using all four processors. The benchmarks were taken using single processor mode.

Figure 4.5 shows the effect of decreasing the maximum resolution levels has on frame rate. These curves were obtained using a horizon of 128 data points (approximately 14 nm). Since 128 may only be reduced six times while maintaining sufficient vertices to generate a mesh at the highest resolution level, frame rate begins to decline after six reductions. Each reduction of maximum resolution requirement gains an extra 3 to 4 frames per second. The effect of Culling shows an extra 2 to 3 frames per second when using five resolutions with a maximum resolution of four.

Figure 4.6 contains a table of measurements of frame rates for various horizon and number of resolution combinations. This table proved useful in developing the AUV *auto_resolution* algorithm. Measurements were taken using a maximum resolution of one (every data point displayed in innermost horizon). These figures were utilized when developing the *auto_resolution()* function for the NPS AUV Simulator. Figure 4.7 provides a view of the filled terrain from an observer perspective.

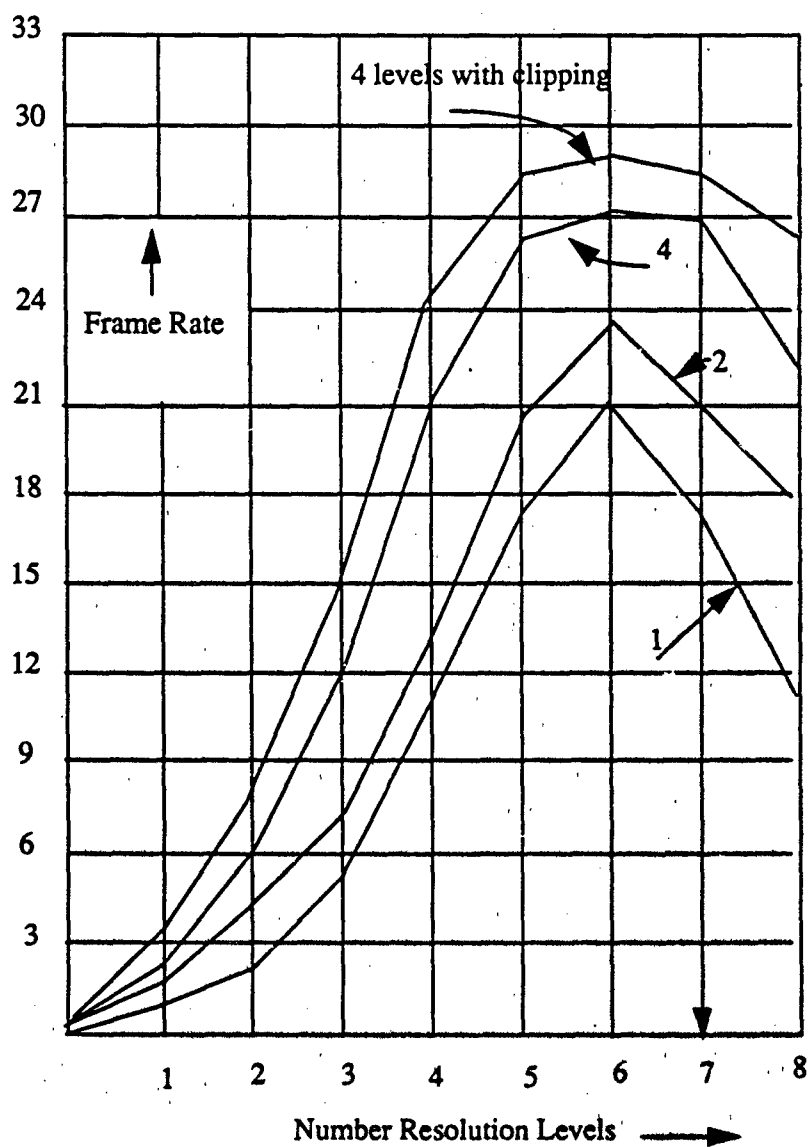


Figure 4.5 Varying the Max Resolution

# Res Horizon	1	2	3	4	5	6	7	8
16	19.1	19.6	23.5					
32	11.1	15.2	19.6	23.5				
64	3.8	6.7	11.5	17.0	19.6			
128	1.6	2.8	5.6	11.1	17.1	21.0		
256	1.0	1.2	2.3	5.5	10.1	15.1	17.0	
512	0.6	0.8	1.1	2.5	5.2	9.1	14.2	16.1
1024	0.2	0.3	0.6	1.1	2.3	4.9	8.9	10.1

By using the formula

$$Horizon = 2^{\text{Number_Resolution_Levels}-1}$$

a frame rate of approximately 10 may be achieved for most horizons.

Figure 4.6VTRA Performance Figures

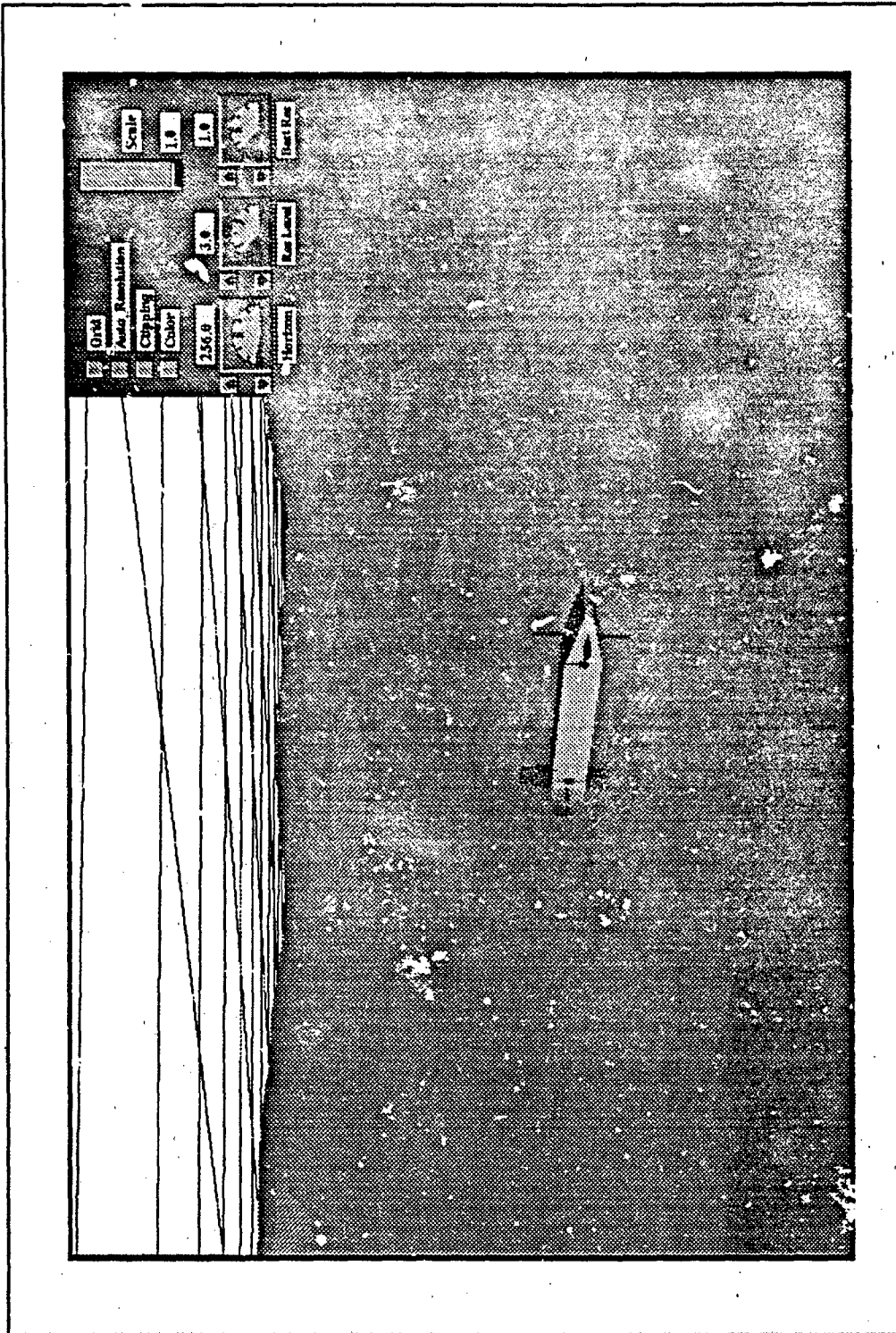


Figure 4.7 Monterey Bay Canyon

V. AUV DATA STRUCTURE

A. INTRODUCTION

When developing the NPS AUV Simulator, the goal was to incorporate object oriented features into the data structure. The auv data structure inherits its characteristics through the use of "C" language pointers.

The auv structure contains five pointers to substructures. The *AUV_Polygons* structure "poly" contains pointers to the vehicle's polygons encapsulated in the NPGS "OFF" format (Munson 1989). The *Dynamic_Structure* "dyn" contains slots required to compute polygon (vertex) transformations from accelerations. The *Vehicle_Geometry* structure "geo" contains information relative to a specific vehicle such as the mass matrix. Information in the mass matrix is used to compute the vehicle accelerations from the vehicle forces. The *Surfaces* structure "surf" contains slots depicting the state of the vehicle's external control surfaces; rudders, fins, and thrusters. The *Coefficients* structure contains items specific to the submarine such as hydrodynamic coefficients which determine, among other things, how much force a given fin deflection will generate, or how much *added mass* needs to be applied to the vehicle during accelerations. Figure 5.1 depicts the essentials of the auv structure.

```

typedef struct {
    AUV_POLYGONS poly;           /* auv objects (polygons) */
    DYNAMIC_STRUCTURE dyn;       /* forces, torques, accelerations */
    VEHICLE_GEOMETRY geo;       /* vehicle geometry struct */
    COEFFICIENTS coeff;         /* hydro coefficients struct */
    SURFACES surf;              /* fin and prop deflections */
} Submarine *Sub_ptr;

```

Figure 5.1 AUV Data Structure

B. AUV_POLYGONS STRUCTURE

The current NPS Object File Format does not support articulated bodies. Transformation may only be applied to the entire object, although work is currently underway to add this capability. The AUV has fifteen separate moving objects; six propellers, eight fins, and the hull. The submarine can be built from the seven OBJECTS contained in the AUV_Polygons structure depicted in Figure 5.2. These OBJECTS are parsed into the structure from an external file at program initialization.

```

typedef struct {
    OBJECT *hullobj; /* ptr to hull polygons */
    OBJECT *stern_planeobj; /* ptr to stern polygons */
    OBJECT *bow_planeobj; /* ptr to bow polygons */
    OBJECT *rudobj; /* ptr to rudder polygons */
    OBJECT *left_propobj; /* ptr to l_prop polygons */
    OBJECT *rt_propobj; /* ptr to r_prop polygons */
    OBJECT *thrusterobj; /* ptr to thruster polygons */
} AUV_POLYGONS;

```

Figure 5.2 AUV Polygons Structure

2. DYNAMICS STRUCTURE

The dynamics structure contains all the essential items for computing the vehicle's dynamics and kinematics, which is more thoroughly covered in the next chapter. However, a brief explanation of some of the slots in Figure 5.3 will be covered here.

```
typedef struct {  
    float delta_time;           /* time between updates */  
    float forces[6];            /* forces & moments */  
    float mminv[6][6];         /* inverse mass matrix */  
    float accelerations[6];     /* udot, vdot, wdot */  
    float velocity[6];         /* u,v,w,p,q,r */  
    float position_change[6];  
    float incremental_H_matrix; /* for body axis rotation */  
    float H_matrix[6][6];      /* rotations and translations */  
    float T_matrix[6][6];      /* for Cockpit View */  
    float pitch, heading, roll; /* phi, theta, psi */  
} DYNAMIC_STRUCTURE;
```

Figure 5.3 Dynamic Structure

1. Delta_time

In AUV1 and AUV2, delta_time was set at 0.5, regardless of the system frame rate. To portray accurate real time dynamics, the system clock must be used. AUV III records the delta time just before each swapbuffer and may be less than .05 (over 20 frames per second). The value is utilized when integrating the accelerations and velocities. The function *new_delta_time()* returns the time (float seconds) since the function was previously called.

```

#include "sys/times.h"
#include "sys/param.h"

float new_delta_time()
{
    struct tms spot_time;      /* structure from times.h */
    static float oldtime;
    float newtime;
    static int timestarted = False;
    float time_difference;

    /* convert clock ticks to seconds using HZ */
    newtime = (float)times(&spot_time)/(float)HZ;

    if (!timestarted) {        /* first reading will be set to zero */
        oldtime = newtime;
        timestarted = True;
    }
    time_difference = newtime-oldtime;
    oldtime = newtime;
    return (time_difference);
}

```

Figure 5.4 Delta Time Routine

2. Forces

This slot contains an array of the forces and moments produced from the equations of motions, which are more thoroughly discussed in the next chapter. The force vector is multiplied by the inverse mass matrix to obtain the accelerations.

3. Inverse Mass Matrix

The inverse mass matrix is determined at program initialization after the mass matrix has been created. The mass matrix is inverted through Gaussian elimination techniques in AUV III. In the previous AUV Simulators, the inverse mass matrix for the SDV was formulated using a Fortran program and then hard coded into the simulator code. The Gaussian elimination routine is shown in Figure 5.5.

```

#define scan(var, lower, upper) for(var=lower; var<upper; var++)
matrix_inverse2(IN_MATRIX, INVERTED_MATRIX, SIZE)
float *IN_MATRIX;
float INVERTED_MATRIX[10000];
int SIZE;

{
    int index=0, row=0, col=0, current_row=0;/
    float **TEMP, factor, row_factor;

    TEMP = (float**)malloc(SIZE * sizeof(float**));          /* allocate memory */
    scan(row,0,SIZE) {                                       /* create temporary augmented matrix */
        TEMP[row] = (float *) malloc(2 * SIZE * sizeof(float));
        scan(col,0, SIZE) {
            TEMP[row][col] = IN_MATRIX[row * SIZE + col];
            TEMP[row][col + SIZE] = 0.0;
        }
        TEMP[row][row + SIZE] = 1.0;
    }
    scan(index,0, SIZE) {                                   /* set factor to diagonal component */
        factor = TEMP[current_row][current_row];
        if (factor == 0) {                                  /* to avoid division by a factor of "0" */
            scan(row, 0, SIZE) {                            /* find row that doesn't have zero */
                if (TEMP[row][current_row] != 0) {
                    scan(col,0, (2*SIZE)) {
                        TEMP[current_row][col] += TEMP[row][col];
                    }
                    factor = TEMP[current_row][current_row];
                    break;
                }
            }
        }
        scan(col,0, (2 * SIZE)) {                          /* divide current row by factor to get a "1" */
            TEMP[current_row][col] = TEMP[current_row][col] / factor;
        }
        scan(row,0, SIZE) {                                 /* subtract (factor * current row) from each row */
            if (row != current_row) {
                row_factor = TEMP[row][current_row];
                scan(col, 0, (2 * SIZE)) {
                    TEMP[row][col] -= row_factor * TEMP[current_row][col];
                }
            }
        }
        current_row++;
    }
    scan(row,0,SIZE) {                                     /* create matrix */
        scan(col,0, SIZE) {
            INVERTED_MATRIX[row*SIZE + col] = TEMP[row][col+SIZE];
        }
    }
}

```

Figure 5.5 Matrix Inverse Routine

4. H-matrix

The homogeneous transform matrix (H-matrix) contains the information required to perform polygon transformations. One format for the H-matrix is shown in figure 5.6 which results from a yaw followed by pitch, then a roll. The transpose of this particular matrix for robotic applications is found in (Fu 87)

$$\begin{bmatrix} C\phi C\theta & S\phi C\theta & -S\theta & 0 \\ C\phi S\theta S\psi - S\phi C\psi & S\phi S\theta S\psi + C\phi C\psi & C\theta S\psi & 0 \\ C\phi S\theta C\psi + S\phi S\psi & S\phi S\theta C\psi - C\phi S\psi & C\theta C\psi & 0 \\ dx & dy & dz & 1 \end{bmatrix}$$

$\phi \rightarrow \text{roll}$
 $\theta \rightarrow \text{pitch}$
 $\psi \rightarrow \text{yaw}$

$dx \rightarrow X\text{position}$
 $dy \rightarrow Y\text{position}$
 $dz \rightarrow Z\text{position}$

Figure 5.6 Homogeneous Transform Matrix

By loading the vehicle's H-matrix onto the transformation stack prior to drawing the vehicle, proper vehicle orientation results. In Chapter 6, we see several uses of the H-matrix in graphics applications. An incremental H-matrix is obtained from incremental rotations and translations using the Graphics Library functions calls *rotate()* and *translate()*. The IRIS 4D uses a right hand coordinate system with X axis left to right, Y axis bottom to top, and negative Z axis into the screen. The vehicle coordinate system is shown in Figure 5.7. To utilize OFF objects on the IRIS with minimum confusion, the vehicle coordinate system should be initially aligned with the IRIS world coordinate system.

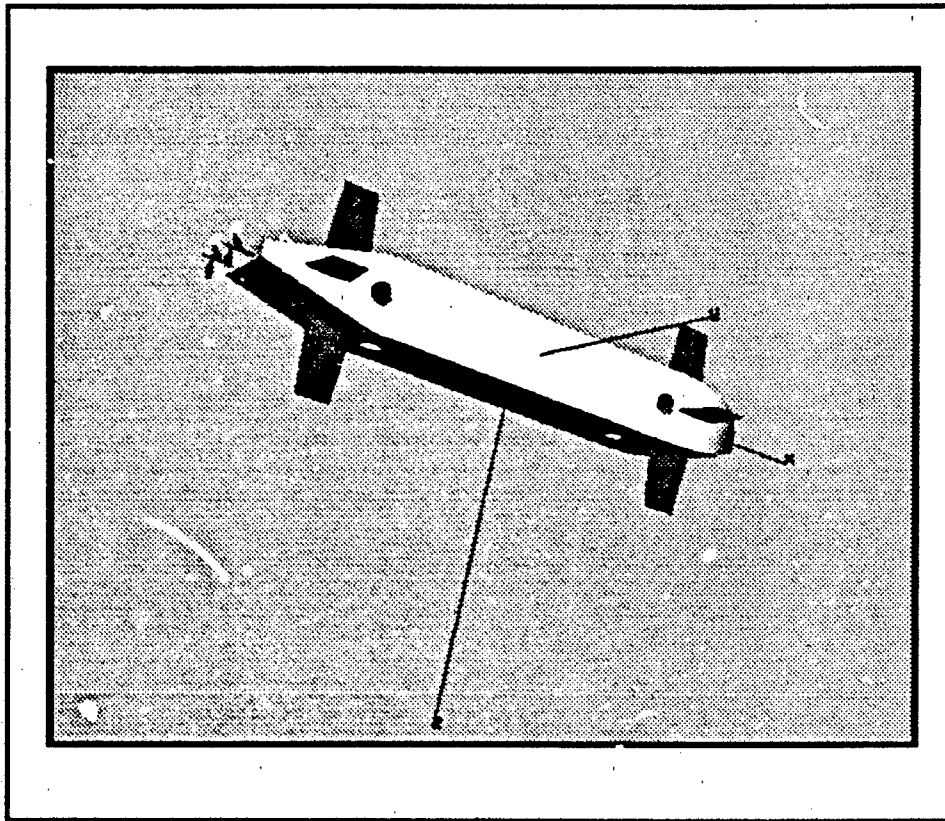


Figure 5.7 Vehicle Coordinate System

5. T-matrix

It is often useful to switch to a "Cockpit" or "Camera" view while operating a vehicle. The function *transpose_matrix()* creates a "T-matrix" from the H-matrix by transposing the upper left 3 x 3 rotation sub-matrix, and by reversing the signs of the translations. An examination of the H-matrix reveals that such a transposition produces rotations opposite that of vehicle rotations. When flying straight and level, a bank to the right has the effect of tilting the horizon to the left.

D. VEHICLE GEOMETRY STRUCTURE

The Vehicle Geometry structure contains information that is peculiar to that type of vehicle. For example, center of buoyancy information is used for ships whereas wingspan area may be used for aircraft. Slots within this structure (Figure 5.8) are amplified in the following paragraphs.

```
typedef struct {  
    float mass;          /* weight / gravity */  
    float weight;        /* to compute mass */  
    float buoyancy;      /* will equal gravity */  
    float length;        /* dimensions in feet */  
    float slice[3][9];   /* length, width, height xsection */  
    float A0;            /* prop area */  
    float xg, yg, zg;    /* distance cg from rotation axis */  
    float xb, yb, zb;    /* distance cb from rotation axis */  
    float ix, iy, iz;    /* symmetric moments of inertia */  
    float ixy, ixz, iyz; /* asymmetric moments of inertia */  
    float mm[6][6];      /* mass matrix */  
} VEHICLE_GEOMETRY;
```

Figure 5.8 Geometry Structure

1. Mass Matrix

The mass matrix is a function of mass, added mass coefficients, center of gravity (xg, yg, zg), moments of inertia (ix, iy, iz), products of inertia (ixy, ixz, iyz), water density(ρ), and vehicle length. At program initiation, the added mass coefficients are read into the added mass coefficient array (see Coefficient Structure below). Using these coefficients and the other aforementioned parameters, the mass matrix is created by the function *compute_mass_matrix()* using the mass matrix equations outlined in (Boncal 1987). The mass matrix was "hard coded" into AUV2 using the SDV parameters. In AUV III, the

parameters are read in at program initialization so the simulator can be run with any geometry. In addition, the AUV III parameters have been modified to reflect the submarines symmetry and size.

2. Other AUV Geometry Considerations

The propeller cross-sectional area, A_0 , is used in the AUV's equations of motion. Since the AUV is considered a rigid body, the various forces can be represented as three discrete forces along the vehicle axes, and the moments as three discrete moments around these axes. These equations are discussed in Chapter 6.

In AUV III, the buoyancy and weight are of equal magnitude. The greater the distance between the center of gravity and center of buoyancy, more stable but less maneuverable, is the submarine. Using *dynamic constraints* as discussed in (Barzel, Barr 1988), one may implement constraints by introducing forces into the model to simulate actual vehicle behavior. In AUV III, by artificially increasing the weight as the vehicle broaches the surface, the equations of motion generate a pitch down motion followed by a vehicle levelling out, preventing a "flying" AUV.

The "slice" matrix is used in the dynamics package to help compute cross flow drag. Nine cross sectional measurements were taken of the AUV for the matrix. Cross flow drag is integrated over the length of the vehicle using the trapezoidal rule with respect to height or width.

E. COEFFICIENTS STRUCTURE

The Coefficients structure is shown in Figure 5.9.

```

typedef struct {
    float surge[6][6];           /* vehicle x axis movement */
    float sway[6][6];           /* vehicle y axis movement */
    float heave[6][6];          /* vehicle z axis movement */
    float roll[6][6];            /* angular about z axis */
    float pitch[6][6];           /* angular about x axis */
    float yaw[6][6];             /* angular about y axis */
    float added_mass[6][6]; /* added mass effects during acceleration */
    float fin_surge[6][4];       /* fin movement & vehicle movement */
    float fin[6][4];             /* fin only */
    char *surge_variables[6][6]; /* required for file regenerations */
    char *sway_variables[6][6];
    char *heave_variables[6][6];
    char *roll_variables[6][6];
    char *pitch_variables[6][6];
    char *yaw_variables[6][6];
    char *added_mass_variables[6][6];
    char *fin_surge_variables[6][4];
    char *fin_variables[6][4];
} COEFFICIENTS, *CO_PTR;

```

Figure 5.9 Coefficient Structure

While accurate hydrocoefficients are often obtained after exhaustive tow tank experiments, the SDV coefficients were obtained through geometrical analysis (Smith 1978). In AUV III, the coefficients were modified to exclude the effects of the third propeller and large skeg on the SDV. Again, in AUV2, the coefficients were "hard coded" and included only those coefficients thought to be affecting the AUV. In AUV III, a full set

of coefficients are included enabling configuration changes (size, control surfaces, propellers, etc.) to be immediately run on the simulator.

Coefficient files can be loaded into the simulator, modified on-line, tested, and saved for reuse. While the added mass coefficients were utilized in the development of the mass matrix, the remaining coefficients are utilized in the equations of motion. Since the AUV is somewhat symmetrical, most of the off-diagonal terms should be approximately zero. Figure 5.10 shows the on-line panel for modifying pitch coefficients. The coefficient can be decoded using the following:

X = Surge

Y = Sway

Z = Heave

K = Roll

M = Pitch

N = Yaw

For example, *MUV* is the coefficient that determines how much pitch force is induced when the vehicle undergoes a surge with a sway. The coefficients should be modified after each in-water testing of the vehicle.

The coefficient files are similar to ordinary "C" language header files and, in fact, may be hard coded at any time. The data structure records the name of the coefficient so that the file may be properly restructured in this "C" format when saving any changes.

F. SURFACES STRUCTURE

The Surfaces structure contains the status of all controls surfaces of the vehicle. The fin deflections and thruster rpms are inputs to the equations of motion. While the AUV has slots for eight fin deflections, enabling independent fin surface control, the current equations do not support this mode. The bow and stern control surfaces are coupled as are

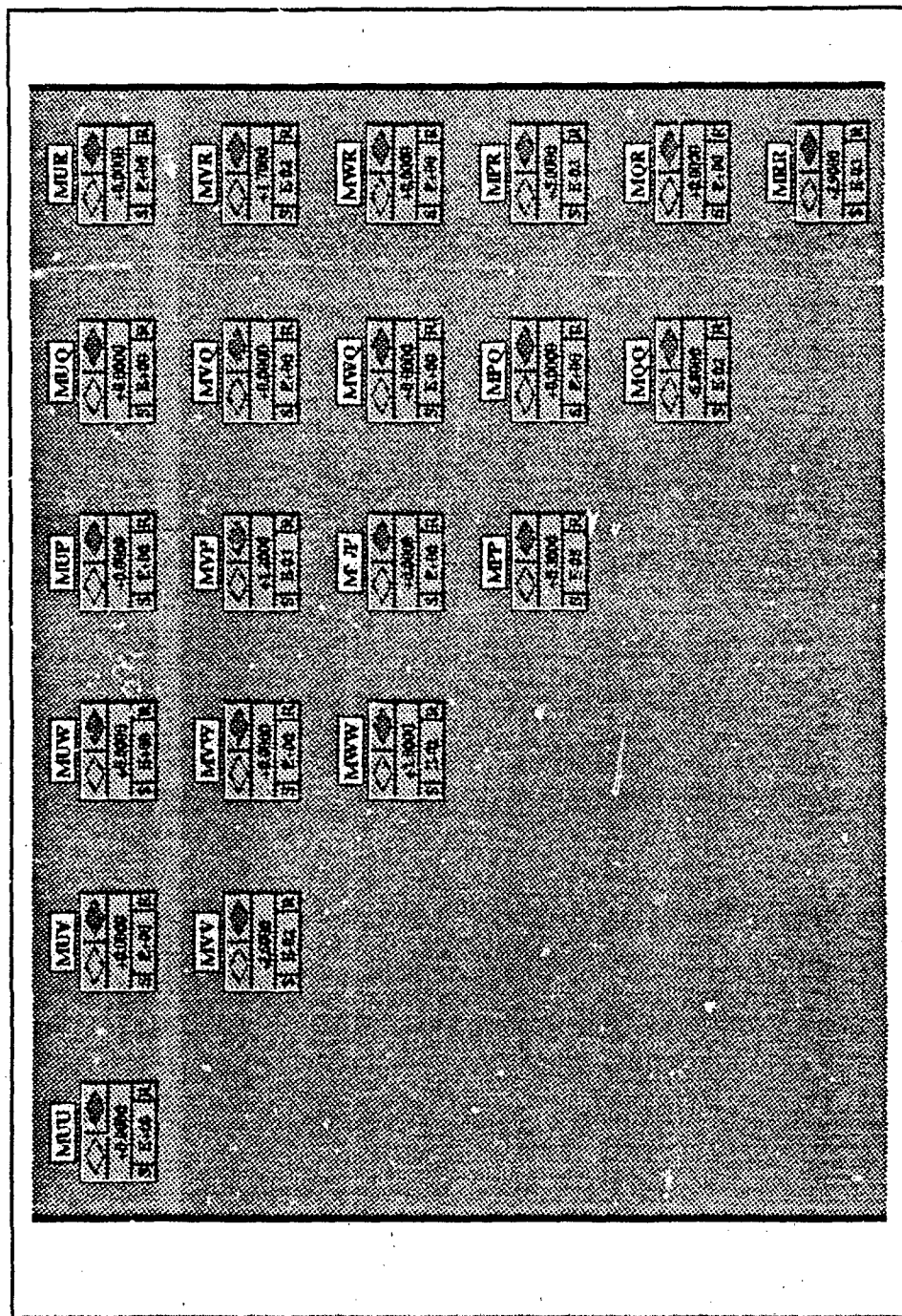


Figure 5.10 Pitch Coefficients Panel

the left and right main thruster. The hovering thrusters are not yet modeled in the equations.

Figure 5.11 depicts the Surfaces structure.

```
typedef struct {  
    float deflect[8];          /* eight control surfaces */  
    float rpm[6];              /* two mains and six thrusters */  
    float prop_disp[6];        /* rotations in one frame */  
} SURFACES;
```

Figure 5.11 Control Surfaces Structure

VI. AUV DYNAMICS

A. INTRODUCTION

1. Dynamics, Animation, and Simulation

With the advent of low cost, powerful graphic workstations, animating rigid body motion through dynamic equations of motion is becoming an attractive alternative to traditional graphics animation techniques such as inverse kinematics, keyframing, and goal directed subsystems (Sturman 1987). While the term "simulation" instead of "animation" suggests a shift of control from the animator to the underlying physics, this need not be the case. The DYNAMO system at Cornell University allows the animator to maintain control of linked figures in the dynamic simulator through use of kinematic constraints and predefined behavior functions (Isaacs 1987). For the animator or "simulator", Sturman suggests that dynamics may be the best way to achieve realistic motion. Jane Wilhelms (Wilhelms 1987) also cites a number of reasons to use dynamics.

- a. Restrict motions to those which are realistic.
- b. Portrays complex motion with minimal user input.
- c. Dynamic constraints can be automatically imposed.
- d. Move complex bodies in a natural way.

2. How to Employ Dynamics

Wilhelms itemizes the steps required to derive object motion from dynamics. Though general in nature, they reflect the procedure used in Cornell's DYNAMO system as well as the NPS AUV Simulator. They are:

- a. Build dynamic equations of motion
- b. Solve equations for forces and accelerations.
- c. Determine velocities and positions through integration.
- d. Update the object's state.

Dynamic constraint checks should be accomplished after step "c". For instance, one constraint may be that the AUV never "fly" out of the water. Rather than using kinematic constraints, i.e., no translations above zero altitude, we would shift the center of buoyancy towards the aft of the vehicle, and then resolve for the equations of motion. The equations would recognize the lower buoyancy moment, and the resultant greater weight moment would cause the vehicle to pitch down into a dive. If the bow and stern planes were not readjusted, there would be a porpoising effect, and the vehicle would remain constrained in its environment.

B. AUV EQUATIONS OF MOTION

The original sets of equations of motion for the AUV were adapted from the submarine equations of motion for the Swimmer's Delivery Vehicle (Boncal 1987). Modifications to the equations included (1) integral formulation of viscous crossflow forces and moments; (2) decoupling of the bow and stern planes (3) decoupling of the left and right bow planes. In AUV III, the viscous cross flow formula was remodified and the third (off axis) propeller was removed.

1. Viscous Crossflow Forces

In order to compute the viscous flow components, nine cross-slice measurements were taken of the AUV. Crossflow components were then calculated by integrating the calculations over the length of the vehicle as shown in Figure 6.1.

```

#define x9 auv->geo.slice[0] /* nine auv height, width measurements */
#define br auv->geo.slice[1]
#define hh auv->geo.slice[2]
#define num_pts 9
#define swayterm (VV + x9[k] * RR)
#define heaveterm (WW - x9[k] * QQ)

compute_drag_force(auv)
Sub_ptr auv;
{
    int k;
    float cyflow, czflow, ucf[num_pts], vech1[num_pts], vech2[num_pts]
        trapezoidal(), vecv1[num_pts], vecv2[num_pts];

    for (k=0; k<num_pts; k++) {
        ucf[k] = fsqrt((swayterm * swayterm) + (heaveterm * heaveterm));
        if(ucf[k] >= 1.e-10) {
            cyflow = f_cdy * hh[k] * swayterm * swayterm;
            czflow = f_cdz * br[k] * heaveterm * heaveterm;
            vech1[k] = (czflow + cyflow) * swayterm / ucf[k];
            vecv1[k] = (czflow + cyflow) * heaveterm / ucf[k];
            vech2[k] = vech1[k] * x9[k];
            vecv2[k] = vecv1[k] * x9[k];
        } else {
            f_heave = f_pitch = f_sway = f_yaw = 0;
            return;
        }
    }

    f_heave = (trapezoidal(num_pts, vecv1, x9)*f_rho/2.)* (-1);
    f_pitch = (trapezoidal(num_pts, vecv2, x9)*f_rho/2.);
    f_sway = (trapezoidal(num_pts, vech1, x9)*f_rho/2.)* (-1);
    f_yaw = (trapezoidal(num_pts, vech2, x9)*f_rho/2.)* (-1);
} /* end compute drag force */

float trapezoidal(points, vel_array, distance)
int points; float vel_array[], distance[];
{
    float answer; int i, j, k;
    j = points; answer = 0.0;
    for (i=0; i<j-1; i++) {
        answer += (.5 * ((vel_array[i]+vel_array[i+1]) *
            (distance[i+1] - distance[i])));
    }
    return (answer);
}

```

Figure 6.1 Viscous Drag Force and Moments

2. Equations Format

The Submarine equations have a standard format as shown in (Boncal 1987).

Figure 6.2 is the header file used to maintain the format while using the AUV data structure.

Figures 6.3 through 6.9 are the current AUV equations.

```
#define f_L    auv->geo.length
#define UU    auv->dyn.vel[0]
#define VV    auv->dyn.vel[1]
#define WW    auv->dyn.vel[2]
#define PP    auv->dyn.vel[3]
#define QQ    auv->dyn.vel[4]
#define RR    auv->dyn.vel[5]
#define phi    auv->roll/57.3
#define theta  auv->pitch/57.3
#define psi    auv->heading/57.3
#define f_rho  1.94 /* auv-> rho    density of water */

/* non-dimensionalized coefficients for use with equations of motion */
#define ndc5(float)(f_rho/2 * f_L * f_L * f_L * f_L * f_L)
#define ndc4(float)(f_rho/2 * f_L * f_L * f_L * f_L)
#define ndc3(float)(f_rho/2 * f_L * f_L * f_L)
#define ndc2(float)(f_rho/2 * f_L * f_L)

#define f_cdy (float)1.0 /* drag factors */
#define f_cdz (float)1.0
#define f_nu (float).00847 /* auv-> nu */
#define f_re (float)(UU * f_L / f_nu)
#define f_eta(.008 * f_rpm / UU) /* vice .012 */
#define f_ct1.008 * f_L * f_L / a0
#define f_ct.008 * f_L * f_L * f_eta * fabs(f_eta) / a0
#define f_cd0(float)(.00085 + (1.296e-17) * (f_re - 1.2e7) * (f_re - 1.2e7))
#define fx_prop(f_cd0 * (f_eta * fabs(f_eta) - 1.0))
#define fn_prop 0
#define fk_prop 0

typedef struct { /* structure used to decompose equations */
    float Newton_Euler[6];
    float hydro_angular_angular[6];
    float hydro_linear_angular[6];
    float hydro_linear_linear[6];
    float drag[6];
    float hydro_static[6];
} Total_Forces, *TForce;
```

Figure 6.2 Equations of Motion Header

```

compute_surge_force(auv,f)
Sub_ptr auv;
TForce f;
{
    int factor1, factor2, acceleration_factor;

    acceleration_factor = fx_prop * UU * UU * f_L * f_L * f_rho / 2.0;

    f->Newton_Euler[X] =
        f_mass * (VV * RR - WW * QQ /* inertial effects */
        + f_xg * (QQ * QQ - RR * RR) /* center of mass effects */
        - f_yg * PP * QQ
        - f_zg * PP * RR);

    f->hydro_angular_angular[X] = /* surge force due to */
        ndc4 * (xpp * PP * PP /* roll */
        + xqq * QQ * QQ /* pitch */
        + xpr * PP * RR /* yaw & roll */
        + xrr * RR * RR); /* yaw */

    f->hydro_linear_angular[X] = /* surge force due to */
        ndc3 * (xwq * WW * QQ + /* heave & pitch */
        + xvp * VV * PP /* sway & roll */
        + xvr * VV * RR); /* sway & yaw */

    f->hydro_linear_linear[X] = /* surge force due to */
        ndc2 * (xvv * VV * VV /* sway */
        + xww * WW * WW /* heave */
        + xvdr * UL * VV * f_dr /* rudder & sway & surge */
        + xwds * UU * WW * f_ds /* stern plane & surge & heave */
        + xwdb * UU * WW * f_db /* bow plane & surge & heave */
        + xqds * UU * QQ * f_ds /* stern plane & surge & pitch */
        + xqdb * UU * QQ * f_db /* bow plane & surge & heave */
        + xdsds * UU * UU * f_ds * f_ds /* stern plane & surge */
        + xdbdb * UU * UU * f_db * f_db /* bow plane & surge */
        + xdrdr * UU * UU * f_dr * f_dr /* rudder & surge */
        + acceleration_factor); /* rpm & surge */

    f->drag[X] = 0.0; /* cross flow drag */

    f->hydro_static[X] = /* buoyancy, weight, pitch angle */
        (-f_weight + boy) * SIN_THETA;
}

```

Figure 6.3 Surge Equation of Motion

```

compute_sway_force(auv,f)
Sub_ptr auv;
TForce f;
{
f->Newton_Euler[Y] = /* sway force due to */
f_mass * (WW * PP - UU * RR /* inertial effects */
- f_xg * PP * QQ /* center of mass effects */
+ f_yg * (RR * RR + PP * PP)
- f_zg * QQ * RR);

f->hydro_angular_angular[Y] = /* sway force due to */
ndc4 * (ypq * PP * QQ + /* roll & pitch */
yqr * QQ * RR); /* pitch & yaw */

f->hydro_linear_angular[Y] = /* sway force due to */
ndc3 * (f_yp * UU * PP + /* surge & roll */
f_yr * UU * RR + /* surge & yaw */
yvq * VV * QQ + /* sway & pitch */
ywp * WW * PP + /* heave & roll */
ywr * WW * RR); /* heave & yaw */

f->hydro_linear_linear[Y] = /* sway force due to */
ndc2 * (f_yv * UU * VV + /* surge & sway */
yv w * VV * WW + /* sway & heave */
ydr * UU * UU * f_dr); /* rudder & surge */

f->drag[Y] =
f_sway; /* viscous left & right cross flow drag */

f->hydro_static[Y] = /* wieght, buoyancy, pitch angle, roll angle */
(f_weight-boy) * COS_THETA * SIN_PHI;
}

```

Figure 6.4 Sway Equation of Motion

```

compute_heave_force(auv,f)
Sub_ptr auv;
TForce f;
{
f->Newton_Euler[Z] = /* heave force due to */
f_mass * (UU * QQ - VV * PP /* inertial effects */
- f_xg * PP * RR /* center of mass effects */
- f_yg * QQ * RR
+ f_zg * (PP * PP + QQ * QQ));

f->hydro_angular_angular[Z] = /* heave force due to */
ndc4 * (zpp * PP * PP + /* roll */
zpr * PP * RR + /* roll & yaw */
zrr * RR * RR); /* yaw */

f->hydro_linear_angular[Z] = /* heave force due to */
ndc3 * (f_zq * UU * QQ + /* surge & pitch */
zvp * VV * PP + /* sway & roll */
zvr * VV * RR); /* sway & yaw */

f->hydro_linear_linear[Z] = /* heave force due to */
ndc2 * (f_zw * UU * WW + /* surge & heave */
zvv * VV * VV + /* sway */
zds * UU * UU * f_ds + /* stern plane & surge */
zdb * UU * UU * f_db); /* bow plane & surge */

f->drag[Z] =
f_heave; /* viscous up/down cross flow drag */

f->hydro_static[Z] = /* buoyancy, weight, pitch, roll angle */
(f_weight-boy) * COS_THETA * COS_PHI;
} /* end heave */

```

Figure 6.5 Heave Equation of Motion

```

compute_roll_moment(auv,f)
Sub_ptr auv;
TForce f;
{
f->Newton_Euler[K] =
(f_iy - f_iz) * QQ * RR - /* moment of inertia effects */
f_ixy * PP * RR + /* product of inertia effects */
f_ixz * PP * QQ +
f_iyz * (QQ * QQ - RR * RR)

+ f_mass * (f_yg * (UU * QQ - VV * PP) - /* center of mass effects */
f_zg * WW * PP + UU * RR);

f->hydro_angular_angular[K] = /* roll moment due to */
ndc5 * (kpq * PP * QQ + /* roll & pitch */
kqr * QQ * RR); /* pitch & yaw */

f->hydro_linear_angular[K] = /* roll moment due to */
ndc4 * (f_kp * UU * PP + /* surge & roll */
f_kr * UU * RR + /* surge & yaw */
kvq * VV * QQ + /* sway & pitch */
kwp * WW * PP + /* heave & roll */
kwr * WW * RR ); /* heave & yaw */

f->hydro_linear_linear[K] = /* roll moment due to */
ndc3 * (f_kv * UU * VV + /* surge & sway */
kvw * VV * WW + /* sway & heave */
fk_prop * UU * UU); /* surge & prop factor */

f->drag[K] = 0.0; /* roll drag effects */

f->hydro_static[K] = /* pitch, roll, buoyancy, weight */
(f_yg * f_weight - f_yb * boy) * COS_THETA * COS_PHI +
(f_zb * boy - f_zg * f_weight) * SIN_PHI;

} /* end roll */

```

Figure 6.6 Roll Equation of Motion

```

compute_pitch_moment(auv,f)
Sub_ptr auv;
TForce f;
{
f->Newton_Euler[M] = /* pitch moment due to */
(f_iz - f_ix) * PP * RR + /* moment of inertia */
f_ixy * QQ * RR - /* product of inertia */
f_iyz * PP * QQ +
f_ixz * (RR * RR + PP * PP)

+ f_mass * (f_xg * (VV * PP + UU * QQ) + /* center of mass effects */
f_zg * (VV * RR - WW * QQ));

f->hydro_angular_angular[M] = /* pitch moment due to */
ndc5 * (mpp * PP * PP + /* roll */
mpr * PP * RR + /* roll & yaw */
mrr * RR * RR); /* yaw */

f->hydro_linear_angular[M] = /* pitch moment due to */
ndc4 * (f_mq * UU * QQ + /* surge & pitch */
mvp * VV * PP + /* sway & roll */
mvr * VV * RR ); /* sway & yaw */

f->hydro_linear_linear[M] = /* pitch moment due to */
ndc3 * (f_mw * UU * WW + /* surge & heave */
mvv * VV * VV + /* sway */
mds * UU * UU * f_ds + /* stern plane & surge */
mdb * UU * UU * f_db); /* bow plane & surge */

f->drag[M] =
f_pitch; /* up/down viscous drag moment */

f->hydro_static[M] = /* buoyancy, weight, pitch, roll*/
(f_xb * boy - f_xg * f_weight) * COS_THETA * COS_PHI
+ ((f_zg - f_zb) * (f_weight-boy)) * SIN_THETA;
} /* end pitch */

```

Figure 6.7 Pitch Equation of Motion

```

compute_yaw_moment(auv,f)
Sub_ptr auv;
TForce f;
{
  f->Newton_Euler[N] = /* yaw moment due to */
  (f_ix - f_iy) * PP * QQ /* moment of inertia */
  + f_ixy * (PP * PP - QQ * QQ) /* products of inertia */
  - f_ixz * QQ * RR
  + f_iyz * PP * RR

  + f_mass * (f_xg * (UU * RR + WW * PP) /* cenert of mass effects */
  - f_yg * (WW * QQ - VV * RR));

  f->hydro_angular_angular[N] = /* yaw moment due to */
  ndc5 * (npq * PP * QQ + /* roll & pitch */
  nqr * QQ * RR); /* pitch & yaw */

  f->hydro_linear_angular[N] = /* yaw moment due to */
  ndc4 * (f_np * UU * PP + /* surge & roll */
  f_nr * UU * RR + /* surge & yaw */
  nvq * VV * QQ + /* sway & pitch */
  nwp * WW * PP + /* heave & roll */
  nwr * WW * RR); /* heave & yaw */

  f->hydro_linear_linear[N] = /* yaw moment due to */
  ndc3 * (f_nv * UU * VV + /* surge & sway */
  nvw * VV * WW + /* sway & heave */
  ndr * UU * UU * f_dr + /* surge & rudder */
  fn_prop * UU * UU); /* surge & prop */

  f->drag[N] =
  -f_yaw; /* left/right drag moment */

  f->hydro_static[N] = /* buoyancy, weight, pitch, roll */
  (f_xg * f_weight - f_xb * boy) * COS_THETA * SIN_PHI
  + (f_yg * f_weight - f_yb * boy) * SIN_THETA;
} /* end yaw */

```

Figure 6.8 Yaw Equation of Motion

C. SOLVING FOR FORCES, TORQUES, & ACCELERATIONS

The six equations (Figures 6.3 - 6.8) are subdivided into six sub-equations. The first, "Newton-Euler", are inertial forces or moments resulting from velocities, moments and products of inertia, as well as force created since the center of mass is not at the center of the AUV's coordinate system (center of buoyancy).

The second set "angular-angular" are forces generated due to rotational velocities around the other two axes. The third set "angular-linear" are forces generated due to a combination of an angular velocity and a linear velocity. The fourth sub-equation solves for the force created when two linear velocities are combined.

The fifth sub-equation shows the force generated due to cross-flow drag as discussed earlier in the chapter. The sixth set are forces due to hydrostatic effects caused by the offset of the center of buoyancy from the center of gravity.

One force or torque is generated for each degree of freedom and placed into a six element force vector. The vector is post-multiplied by the inverse mass matrix to produce a six element acceleration vector (Figure 6.9).

```
/* multiply force vector by inverse mass matrix to get acceleration vector */
compute_accelerations(auv)
Sub_ptr auv;
{
  int j,k;
  for (j=0; j<6; j++) {
    for (k=0; k<6; k++){
      auv->dyn.acc[j] += auv->dyn.mminv[j][k] * auv->dyn.forces[k];
    }
  }
}
```

Figure 6.9 Solving for Accelerations

D. SOLVING FOR VELOCITIES & POSITION CHANGES

There are many integration methods available for deriving positional information from the acceleration vector. One of the most accurate but more complex and slower is the *Runge-Kutta Method* (Wilhelms 1987). On the other end of the spectrum is the *Euler Method* which is fast, but inaccurate, especially for higher delta times. This is illustrated in Figure 6.10.

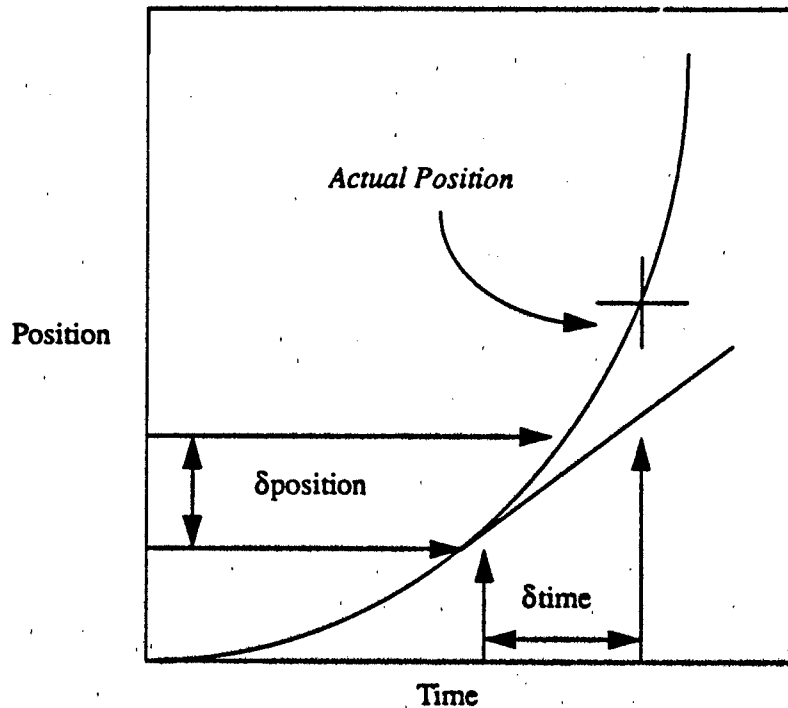


Figure 6.10 Velocity Curve

The *Euler method* samples the velocity at a given point, assumes a constant velocity, and calculates a future position according to the equation below. The calculation

$$p1 = p0 + v0 \cdot \delta t$$

works well for very small delta times, but can be erroneous at higher intervals. During non-

graphical AUV dynamic calculations, where real time is not an issue, the delta times can be very small indeed. Preliminary AUV dynamic tests used delta times of 1 msec. AUV SIM2 needed to approximate the graphical frame rate to get realistic motions and used a delta time of 0.5 seconds in conjunction with the Euler Method. AUV SIM III uses the system time to get an accurate delta time, usually between 0.1 and 0.15 seconds. It was felt that even this interval was too great for the Euler Method, so the *Modified Euler Method* (Spiegel 1988) was adopted. In the modified method, a predicted average velocity, rather than an initial velocity is used. The average velocity is calculated based on the acceleration at sample time, and the equation now becomes:

$$p1 = p0 + v0 \cdot \delta t + (a0 \cdot (\delta t)^2) / 2$$

The routine for calculating the new velocities and position changes is depicted in Figure 6.11.

```
compute_velocities_and_positions(auv)
Sub_ptr auv;
{
  int i;
  static int dtr = 573; /* degree to radian conversion (SGI uses 10ths) */
  for (i=0; i<6; i++) {
    auv->dyn.vel[i] += auv->dyn.delta_t * auv->dyn.acc[i];
    auv->dyn.pos_change[i] =
      ((auv->dyn.delta_t * auv->dyn.vel[i]) +
       (auv->dyn.delta_t * auv->dyn.delta_t *
        auv->dyn.acc[i] / 2.0));
    if (i>2) auv->dyn.pos_change[i] *= dtr;
  }
}
```

Figure 6.11 Velocity and Position Change Routine

E. UPDATING THE AUV'S STATE

1. Create the incremental_H_matrix

The incremental positional changes are relative to the vehicle coordinate system, and must be converted to the world coordinate system. By loading the incremental changes into an incremental homogeneous transform matrix, we can take advantage of the IRIS 4D transformation stack and the graphics library *multmatrix()* function to make the necessary transformations. The routine in Figure 6.12 creates the incremental H-matrix, again taking advantage of the transformation stack and the graphics library functions *rotate()* and *getmatrix()*.

```
void get_incremental_H_matrix_from_position_changes(auv)
Sub_ptr auv;
{
    pushmatrix();
    loadunit();
    rotate(-(Angle)auv->dyn.pos_change[5], 'y'); /* yaw */
    rotate( (Angle)auv->dyn.pos_change[4], 'x'); /* pitch */
    rotate(-(Angle)auv->dyn.pos_change[3], 'z'); /* roll */
    getmatrix(auv->dyn.incremental_H_matrix);
    popmatrix();
}
```

Figure 6.12 Transforming to World Coordinates

a. Rotation Order Matters

In Figure 6.12, notice that the yaw is applied first, and the roll is applied last. Euler Angle Rotations are not commutative, therefore a starting axis must be chosen. The order applied here is the standard order for aeronautical applications, and most readily

1987), and most readily adapts to aircraft motions. This may be because rolls usually have the highest Euler Angle rates, and yaws usually have the lowest rates. We do not wish to have the rolls altered by follow on rotations. Shoemaker makes an argument for the use of quaternions instead of Euler Angles for modeling transformations. With quaternions, rotation *order is not a factor*. Although it is possible to convert between quaternions and matrices, Shoemaker describes such a process as "ill-defined". The quaternion representation for rotations is

$$\text{Rot}(n, \theta)$$

where the final orientation is a rotation of angle theta around a single axis n . (Fu 87). The use of quaternions for the AUV flight simulation is a moot point in that the IRIS 4D software and hardware is based on 4 x 4 transformation matrices.

b. Vehicle Coordinate System Alignment

Further examination of Figure 6.12 reveals that an AUV yaw occurs around the vehicles "z" axis or the "y" axis on the world coordinate system. The difference in the coordinate systems was described earlier in the chapter, but is amplified here. The AUV object was developed external to the NPS Simulator, when called into the program, the vehicle's positive "x" axis is aligned with the world coordinate system's "-z" axis. The vehicles "y" axis is aligned with the world's "x" axis. Although not necessary, it would be less confusing to someone reviewing the code if the vehicle was redesigned to be initially aligned with the IRIS 4D world coordinate system.

2. Revising the Homogeneous Transform Matrix

The incremental transformation is converted to world coordinates in the routine shown in Figure 6.13, where the vehicle's homogenous transformation matrix is updated.

```
void update_H_matrix_from_incremental_changes(auv)
Sub_ptr auv;
{
    pushmatrix();
    loadunit();
    multmatrix(auv->dyn.H_matrix); /* old rotations & translations */
    multmatrix(auv->dyn.incremental_H_matrix)
    translate(auv->dyn.pos_change[1],
              -auv->dyn.pos_change[2],
              -auv->dyn.pos_change[0]);
    getmatrix(auv->H_matrix); /* new rotations & translations */
    popmatrix();
}
```

Figure 6.13 Updating the Vehicle's State

Multmatrix() premultiplies the top of the stack by its argument, with the new value being placed on the stack (Graphics Library 1988). By premultiplying the H-matrix by the incremental H-matrix, we have the net effect of a vehicle rotating around its own axis. If we were to reverse the order, the rotations would occur around the world coordinate system, often used when directly positioning objects on the screen using a virtual reality input device such as a spaceball.

3. Extracting Pitch, Roll, and Heading information

Unlike other graphic simulators at NPS, we have incorporated vehicle translations around all three axes, and have done so without ever tracking pitch, roll, or heading information. The vehicle state was maintained using the viewing matrix, which was coined the "H-matrix" and made part of the AUV structure. Why is it then necessary to extract pitch, roll, and heading information? The primary answer is to supply feedback via the user interface in a format more readily assimilated by the user. A vehicle heading of 045 degrees means more to a user than "H-matrix[0][3] = .707" (which incidently means that the vehicle is *either* heading 045 degrees *or* 315 degrees).

Having pitch, heading, and roll information readily available is helpful in other ways also. When using *dynamic constraints*, we may wish to superimpose a roll limitation on our vehicle. Pitch and heading information may be helpful when utilizing the bow mounted sonar, and simulating contact within the sonar acquisition cone.

Pitch is limited to +/- 90 degrees whereas roll is limited to +/- 180 degrees and heading ranges from 0 to 360 degrees. Pitch can be calculated directly from one of two sine values in the matrix. However, roll and heading, as previously pointed out, may be ambiguous. By utilizing the cosine information in the H-matrix diagonals, this ambiguity can be resolved. The routine in Figure 6.14 shows the extracting process.

```

void extract_heading_pitch_and_roll_from_H_matrix(auv)
Sub_ptr auv;
{
    auv->dyn.heading = -asin(auv->dyn.H_matrix[2][0])*57.3;
    if (auv->dyn.H_matrix[2][2] < 0.0){
        auv->dyn.heading = 180 - auv->dyn.heading;
    } /* heading into "z" */
    if (auv->dyn.heading < 0){
        auv->dyn.heading += 360;
    } /* limit heading to 360 degrees */
    auv->dyn.pitch = -asin(auv->dyn.H_matrix[2][1])*57.3
    auv->dyn.roll = asin(auv->dyn.H_matrix[0][1])*57.3;
    if (auv->dyn.H_matrix[1][1]>0.0){
        auv->dyn.roll = 180 - auv->dyn.roll; /* upside down */
    }
    auv->dyn.roll += 180; /* starting along -z axis */
    if (auv->dyn.roll>180) {
        auv->dyn.roll = auv->dyn.roll - 360;
    } /* limit roll to 360 degrees */
    if (auv->dyn.roll<(-180)) {
        auv->dyn.roll = 360 + auv->dyn.roll;
    } /* limit roll to +/- 180 degrees */
} /* end extract heading ... */

```

Figure 6.14 Pitch, Roll, & Yaw

F. DYNAMICS AND REAL TIME APPLICATIONS

1. Dynamics is not the Limiting Factor

With the computational power of today's high performance graphic workstations, dynamics as a means of simulating motion is much more achievable. While streamlining the equations is important, e.g. lookup tables instead of trigometric function calls, multiplication instead of exponential functions, the extra overhead was remarkably low. Two modes are available with the NPS AUV III Simulator, non-dynamic and dynamic.

a. Dynamic and Non-dynamic Modes

In the non-dynamic mode, all the vehicle transformations occur as a result of direct spaceball or mouse panel inputs to the Homogeneous Transform Matrix. In the dynamic mode, the inputs are sent to the function *dynamics()*. This function computes the cross flow viscosity, solves equations of motion for six axes, performs a matrix multiplication to produce an acceleration vector, performs an Euler integration on the accelerations and a *modified* Euler integration on the velocities, applies the incremental position changes to the incremental_H_matrix which premultiplies the vehicle H_matrix to obtain the revised vehicle state.

b. Dynamic Mode Benchmarks

Benchmarks were obtained to compare the load that the dynamics package had on the system frame rate. In the swimming pool with Cockpit view, the frame rate was 16.2, *with and without* the dynamics package activated. With the AUV displayed along with the pool, the frame rate was 7.7, again, *with or without*, the dynamics package activated. Whereas color presentation, terrain resolution, and even panel interface display had a negative effect on frame rate, the use of the dynamics package had none. My conclusion is that, for this dynamics model, as long as the mathematics is kept outside the

mesh drawing loop, dynamics is an effective way to model the AUV's behavior in real time.

2. Parallel Processing

In anticipation of a heavy system drain due to dynamic equation calculations, the dynamics package was designed to take advantage of the multiple processors of the IRIS workstation. Using *barriers*, the six general equations can be solved simultaneously, halt at the barrier until all equations are solved, the force & torque vector generated and accelerations computed. The design is reflected in Figure 6.15.

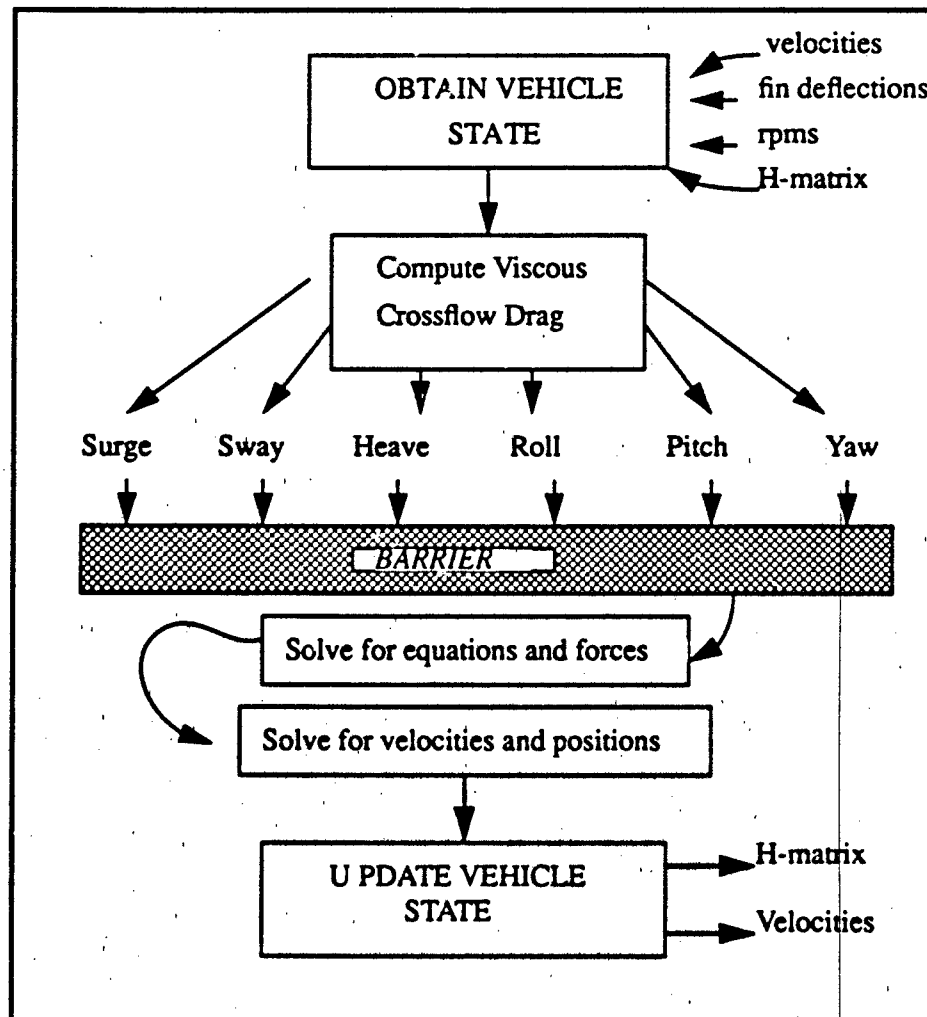


Figure 6.15 Parallel Processing Diagram

In addition to solving the equations in parallel, on a higher level, the dynamics and the graphics could be done in parallel using a producer-consumer model. With the addition of a duplicate H-matrix, the graphics package could lag one frame behind the dynamics package. Although the speed of the dynamics package currently does not warrant the overhead of parallel processing, incorporation of dynamic constraints and collision detection could degrade performance to such a level that co-processing can become an attractive alternative.

3. Addition of Dynamic Constraints

Dynamic constraint checking can be built on to the tail end of the dynamics package. If the vehicle's state fails to abide by the constraints, the variables within the equations of motion could be temporarily changed, and the vehicle's state sent back through the dynamics package. When the constraints are satisfied, control returns to the graphics package.

VII. NPS AUV SIMULATOR

A. USER INTERFACE

The User Interface was generated utilizing the NPS Panel Designer (NPSPD) (King, Prevatt 1990). NPSPD generates "C" code including a primary graphics control loop where the user can place his/her routines. Since the NPS AUV III Simulator was already a working program, the incorporation of a NPSPD Interface had to be reversed engineered. The NPS AUV Simulator, along with the NPS Material Editor, were the first programs to incorporate NPSPD. In the (King, Prevatt) thesis, there is a chapter which describes how the AUV simulator incorporated NPSPD. The main points are covered here for completeness.

The array of panels are contained in the program *viewer.c*, as are some of the panel actuators. As the list of panels grew, it was easier to track and modify if each group of panel actuators were stored in separate files. The files are tied into *viewer.c* using "include" preprocessor statements. For instance, all the actuators on the tape recorder panel are stored in *acuator.dir/recorder.act*, although the recorder panel itself is stored in *viewer.c*.

When a new panel of actuators is generated, the global "MAX_PANELS" in *viewer.h* must be incremented by one. The array number assigned to the panel and actuator array must be one higher than the most recent panel addition. The coefficient panels have the most actuators with 33. If any new panel exceeds that amount, the MAX_ACTUATOR globals in *viewer.h* must be adjusted. The path to the panel library is in the *Makefile*. Whenever new panels are generated, the program must be relinked to the library.

B. MASTER SELECTION PANEL

The Master Selection Panel, shown in Figure 7.1, is composed of two subsections. The push-button panel selections, and the viewing perspective panel.

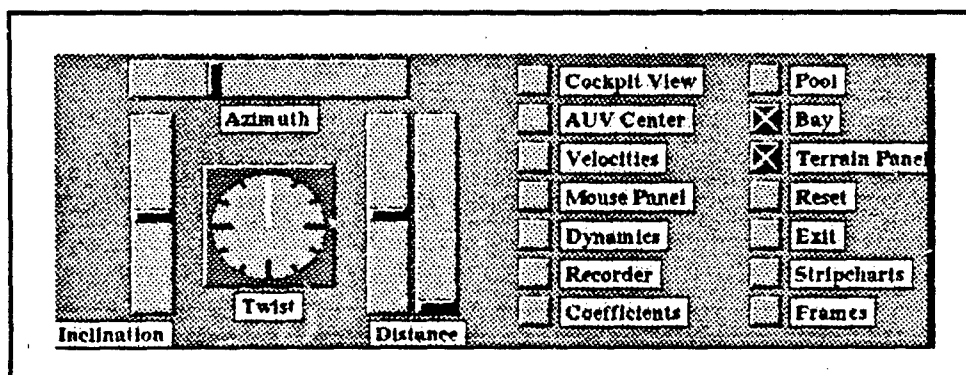


Figure 7.1 Master Selection Panel

The viewing panel controls the "latitude", "longitude", and head "twist" from which the vehicle is viewed. There are two distance bars. The one on the left is for small adjustments using the left mouse. Superfine adjustments are made using two mouse buttons, the left and the center. The right bar is for making coarser adjustment such as getting a "satellite view" of the California Coastline.

The panel select panel primarily activates sub-panels, as well as selects the environment, bay or pool. "Cockpit View" enables the user to view the environment from a nose mounted camera. AUV Center, the default selection, keeps the vehicle in the center of the display; the observer "moves with the vehicle". When deselected, the viewer looks at where the vehicle was when the deselection was made. The vehicle can then actually be flown out of the field of view.

C. MOUSE PANEL

Although manual control is primarily with the spaceball, adjustment to control surfaces or RPM can be made via the mouse panel. This is useful not only when there is no spaceball with the workstation, but also when it is critical to activate one set of controls only, a very difficult accomplishment using the six degree of freedom spaceball. The mouse panel is shown in Figure 7.2

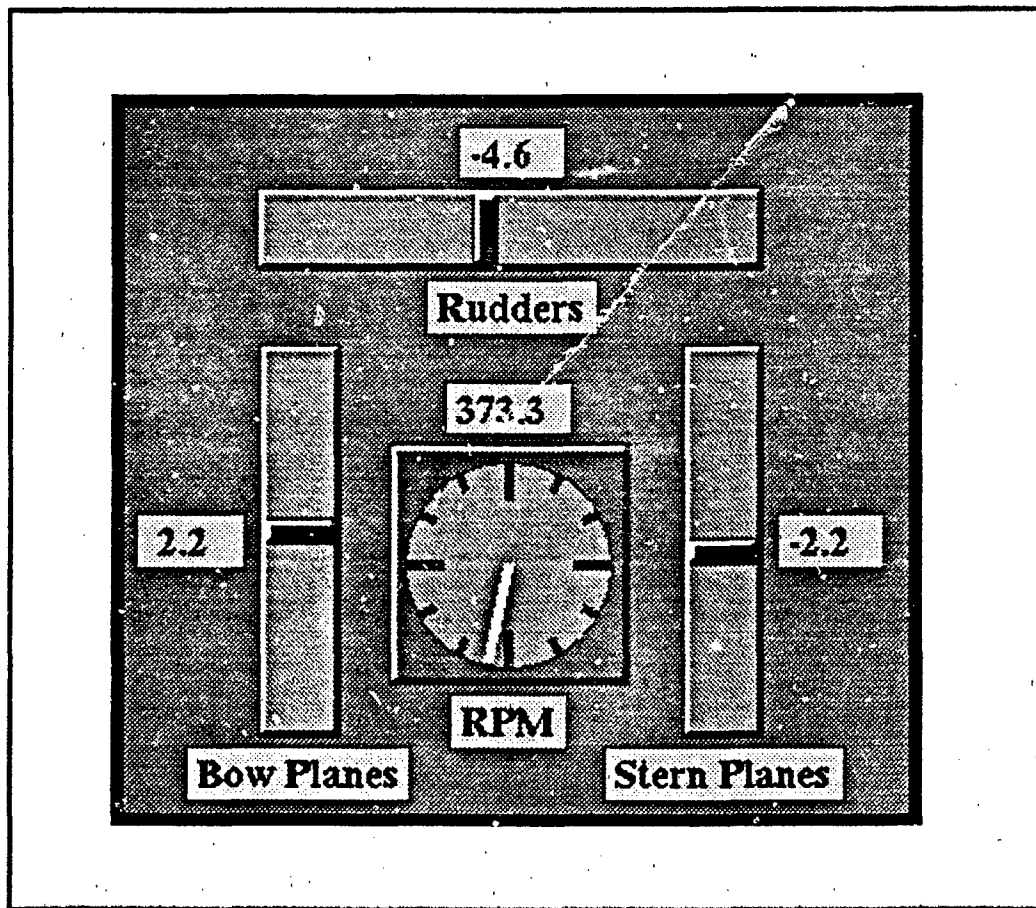


Figure 7.2 Mouse Panel

Currently the Stern Planes can not be activated by the mouse as they are coupled to the bow planes. Rudder Limits are +/- 40 degrees, and RPM limit is 700.

D. PERFORMANCE PANEL

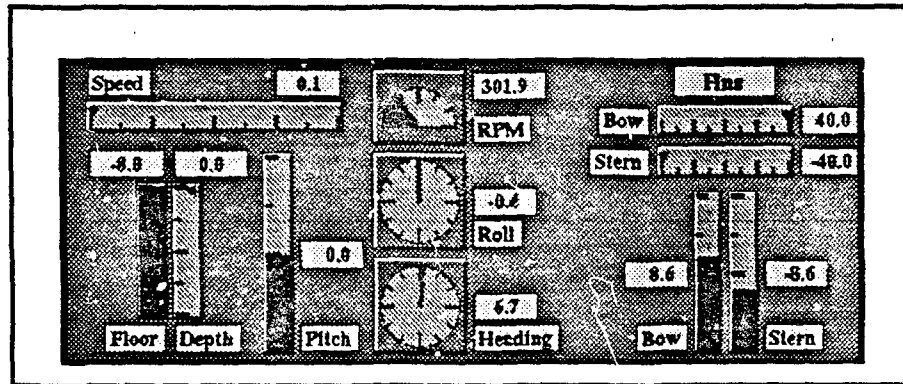


Figure 7.3 Performance Panel

The Performance Panel displays a partial vehicle state. The Speed is calibrated to be in knots. The gauge limit on the RPM is 1000. The Depth meter indicates vehicle depth while the Floor meter shows bottom clearance informations. Future expansion should include a fin deflection meter for all 8 control surfaces, and an RPM dial for each of the six thrusters.

E. FRAMES PANEL

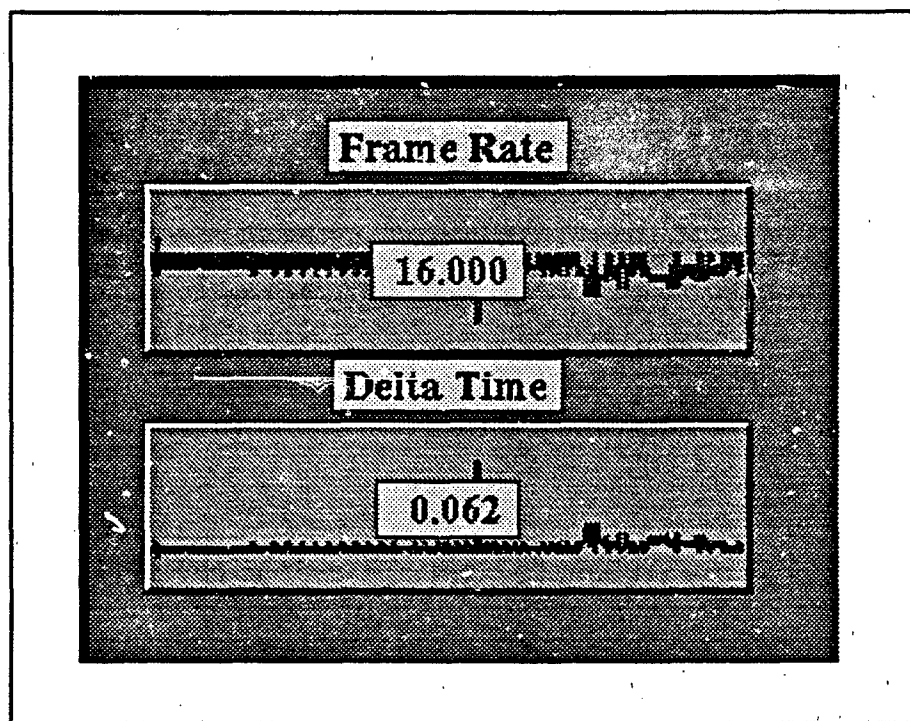


Figure 7.4 Frame Panel

The Frames panel gives the workstation's performance in two ways. Delta Time is the total time between swapbuffers, and frame rate is the inverse, i.e., total frames per second. The meters are single pen stripcharts and are located on the lower left portion of the screen.

F. RECORDER PANEL

The recorder panel provides the capability to record and replay scenarios.

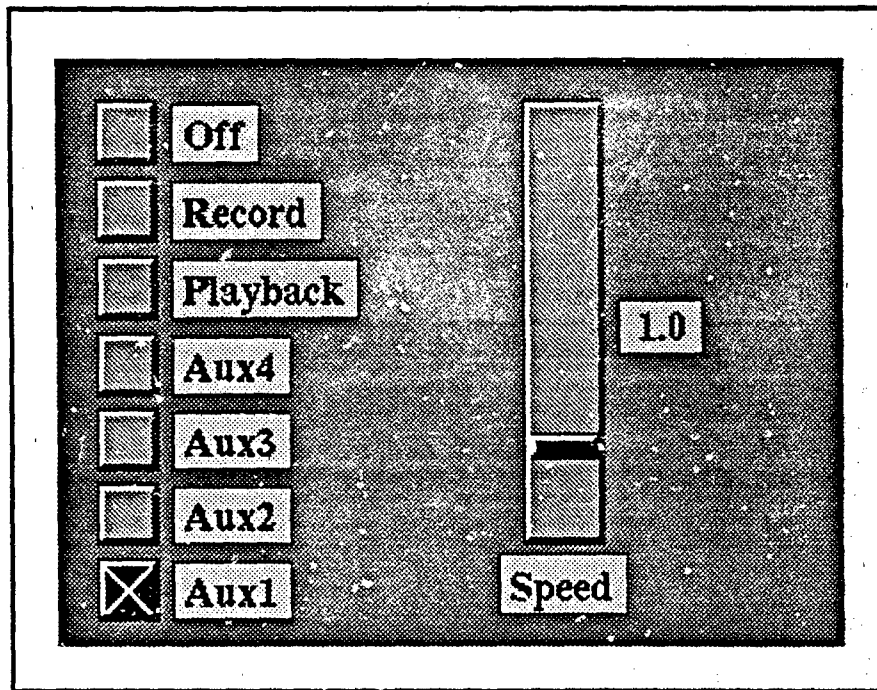


Figure 7.5 Recorder Panel

The recording is made in the ASCII file "recording" which contains the initial vehicle state followed by times, rpms and fin deflection whenever a change of rpm on defelection occurred. When "Record" is selected, it erases the previous tape. Playbacks can occur as many times as desired without erasing the tape. External scripts may be played if they are loaded to the "recording" file. The Tape Speed selection adjusts the speed of playback. The Aux buttons are selectable, and available for programming in the *auv_to_panel_interface.c* package.

G. VELOCITIES PANEL

The velocities panel shows accelerations and velocities for each of the six degrees of freedom.

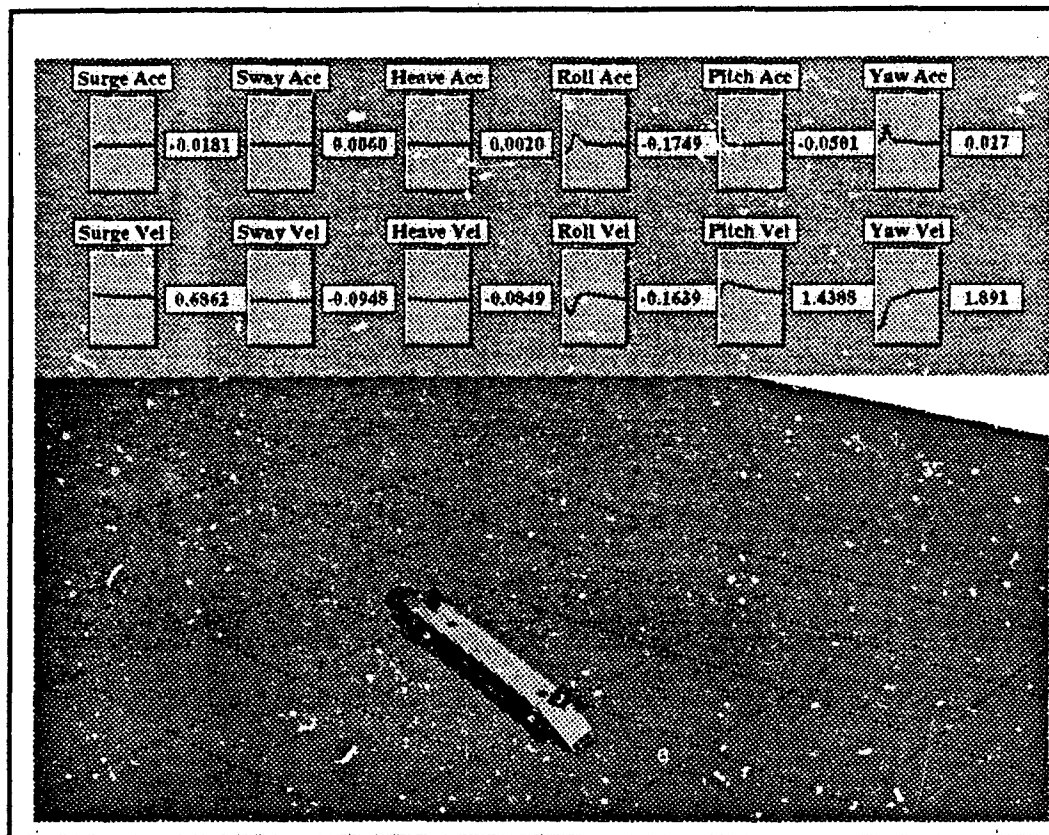


Figure 7.6 Velocities Panel

This panel is utilized to monitor the state of the vehicle while running dynamic tests. Comparisons with data obtained from in-pool testing should reveal whether or not the vehicle is responding appropriately. The acceleration values are in feet/sec/sec, and the velocity values are in feet/sec. To activate the panel, the Stripcharts button must be activated followed by the Velocities button.

H. BOTTOM CONTOUR PANEL

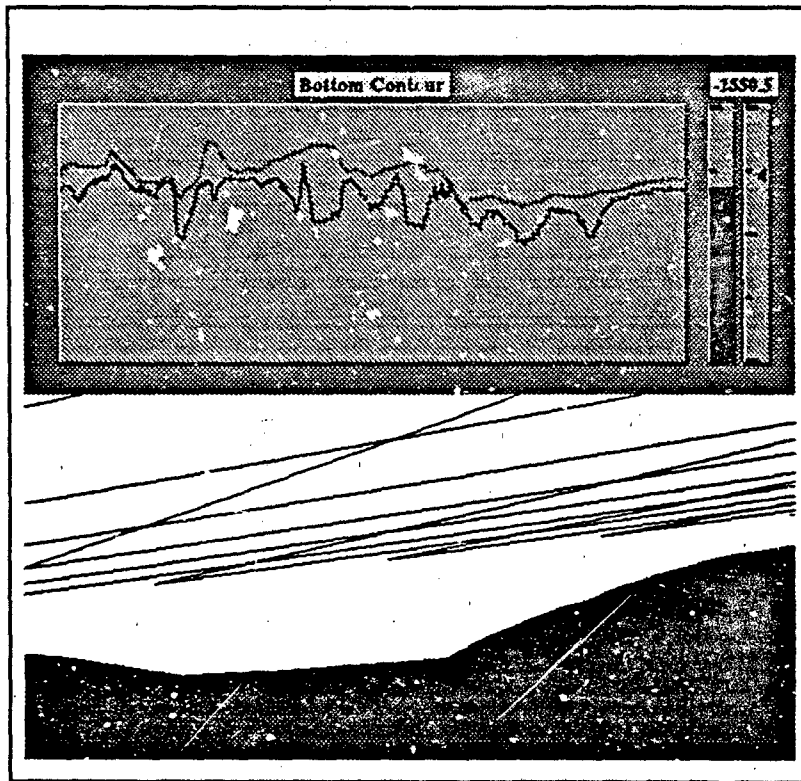


Figure 7.7 Bottom Contour

The bottom contour chart shows a dual pen stripchart that plots both the submarine's depth and the depth of the sea floor. The above chart shows the vehicle in essentially a terrain following mode. The meters on the right repeat the values that are on the stripchart. By default, the upper pen is red, and the lower pen is black. The actuator is located on the upper right portion of the screen.

I. TERRAIN PANEL

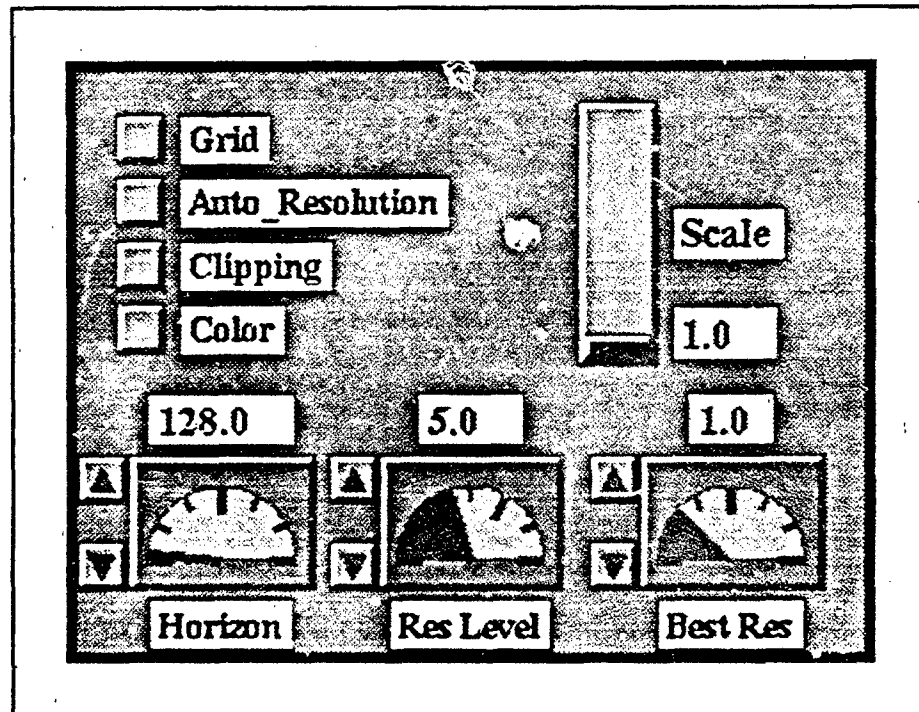
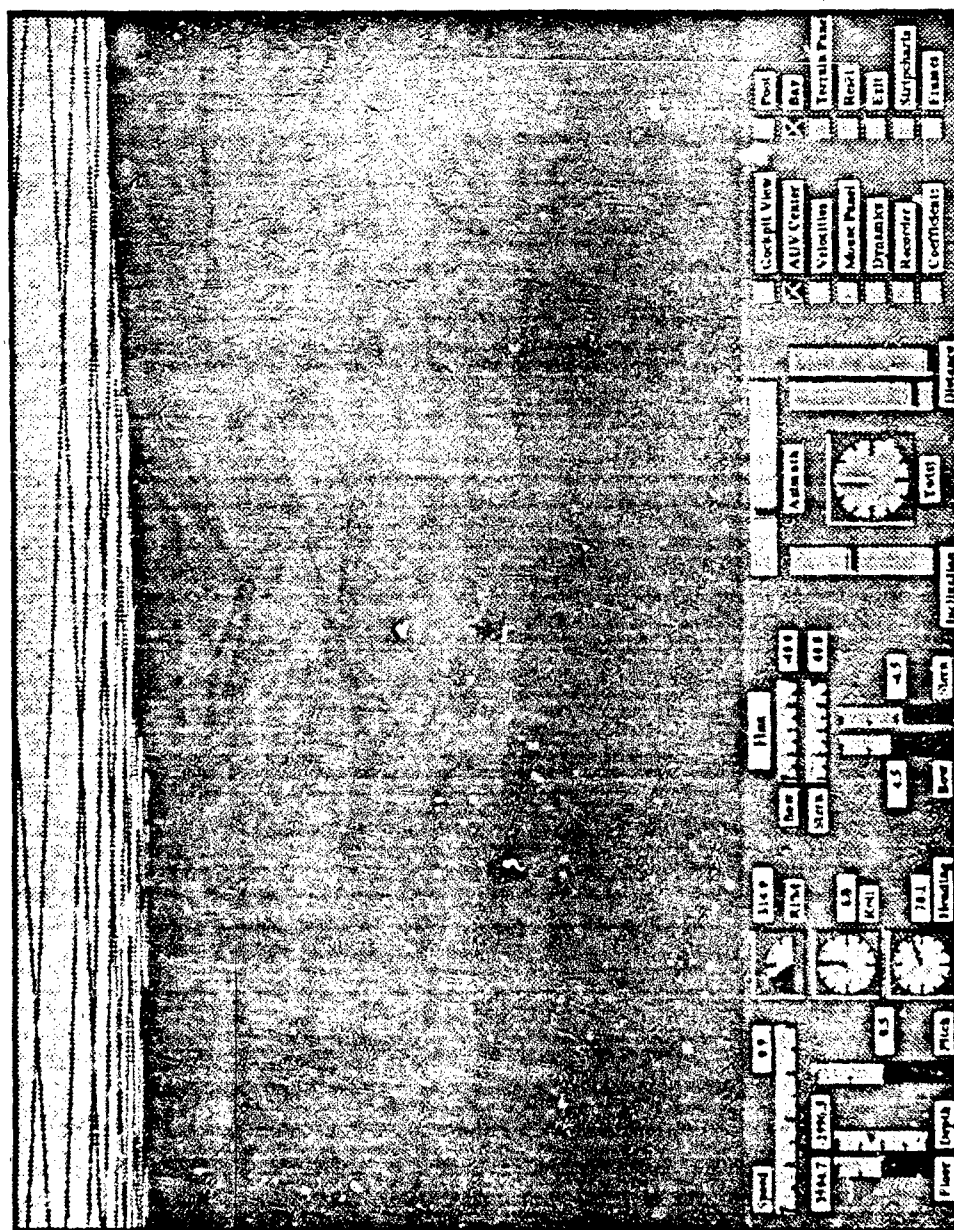


Figure 7.8 Terrain Panel

The terrain panel allows variable terrain resolution selection, variable number of resolution level selection, and density of the best resolution selection. The Grid button deselects filled mode. Clipping activates the culling algorithm to eliminate vertices outside the field of view. Color selects elevation color coded data for display. The scale bar is non-functional but available for programming.



VIII. FUTURE DIRECTIONS

A. DYNAMIC CONSTRAINTS AND PARALLEL PROCESSING

Since the incorporation of dynamic constraint checking could require significant overhead, parallel processing of the NPS AUV Simulator can be advantageous. Constraint checking could cause the input parameters to the equations of motion to be continually adjusted, and the equations resolved in a loop until the constraints are satisfied. Co-processing of the six equations can have a favorable effect. In addition, placing the graphics routine and dynamics/constraints package in parallel can be beneficial as well.

The vehicle broaching the surface can be emulated by adjusting the center of buoyancy and/or the buoyancy vector. Normally the center of buoyancy is assumed to be at the center of rotation, and the buoyancy is equal to the weight.

The proper response of a vehicle collision with the pool wall or floor is dependent on what part of the vehicle makes contact, and the state of the vehicle at time of collision. A single point force on the vehicle will need to be factored into the equations until the constraint is satisfied.

The effect of the vehicle running aground is similar to the pool scenario described above, however, type of bottom, e.g., sand, rock, or silt, would need to be considered.

Since the equations produce approximations of the vehicle behavior, certain input parameter values can potentially display unrealistic or undesired behavior. For example, it may be necessary to set a maximum positive and negative rpm on the main thrusters.

B. INCORPORATION OF PERIPHERAL PACKAGES

1. Controller

The controller (autopilot) package needs to be incorporated. The controller should provide rpm and fin deflection information to the AUV based upon desired heading, pitch,

and speed. Using sliding mode control, the bow planes will act independently of the stern planes. Future controller improvements include separate control mode for all eight surfaces, and a hovering mode using the vertical and horizontal thruster.

2. Navigator

The navigator will compute desired heading, pitch, and speed based upon waypoint data (3D position and time on top). If drift is detected based upon doppler sonar information, this should be included in the computation. If predicted current information is available, that should be included in the "dead reckoning (DR)" process. The DR position can be "fixed" using bottom contour information available in the environmental data base.

3. Mission Planner/Replanner

The mission planner would generate desired waypoints based upon the specific mission of the submarine, e.g. bottom mapping, main countermeasures, special forces support, etc.. The replanner would regenerate waypoints based upon newly available information, e.g. unplanned obstacles, vehicle emergency, enemy detection, revised mission, etc..

C. TERRAIN

The VTRA needs to be expanded to include multiple data cells. Since VTRA supports a grid database, such as that available from DMA, submarine missions can be simulated nearly anywhere worldwide. Based on vehicle's speed, direction, and horizon, adjacent grid cells will need to be transferred from peripheral storage to primary storage automatically. VTRA will need to be enhanced to manage the data transfer, and terrain cell management.

Elevation coded terrain coloring should be incorporated. The elevation data needs to be checked and, if required, a new lighting model generated, for each vertex in the graphics

pipeline. Texturing is a another alternative to displaying realistic terrain as demonstrated on NPSNET, and should be incorporated.

D. AUV MODEL DRAWING

The AUV is currently drawn using polygons and surfaces. By drawing the hull and fins as meshes, vehicle appearance can be maintained while reducing the graphics pipeline load as described earlier. With mesh drawing, an algorithm similar to VTRA could be developed to display the AUV at various resolution levels. The NPS Object File Format (OFF) is currently being revised to incorporate articulated objects. Future redrawing of the vehicle should be based upon the future OFF design. Colors should be carefully chosen so as to be compatible with NTSC displays. RGB warm colors (red, orange, etc.) often get distorted when converted to NTSC, and should be viewed prior to selection using the NPS Material Editor (NPSME) (Anderson 1990).

E. CONCLUSIONS

The Variable Terrain Resolution Algorithm enables terrain grid databases, such as DMA DTED, to be displayed with further horizons and minimal loss of graphic display speed, while maintaining high resolution terrain near the observer. The dynamics of the AUV can be modeled in real time. Hydrodynamic coefficients can be adjusted on line for a quick refinement of the vehicle's performance characteristics. The NPS AUV Simulator can enhance vehicle software development without actual in water trials. Dynamics can be adapted to other NPS Simulators by incorporating the rigid body dynamic model used in the AUV Simulator.

LIST OF REFERENCES

- Smith, Douglas B., and Streyle, Dale G., *An Inexpensive Real-Time Interactive Three-Dimensional Flight Simulation System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1987.
- Oliver, Michael R., and Stahl, David J., *Interactive, Networked, Moving Platform Simulators*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1987.
- Winn, Michael and Strong, Randolph, *Moving Platform Simulator II: A Networked Real-Time Simulator With Intervisibility Displays*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1989.
- Fichten Mark A. and Jennings, David H., *Meaningful Real-Time Graphics Workstation Performance Measurements*, M.S. Thesis, Naval Postgraduate School, Monterey California, 1988.
- Schachter, Bruce J., *Computer Image Generation*, John Wiley & Sons, Inc., 1983, pp. 138-140.
- Harris, F., Yurchak J., Zyda. M., *Preliminary Work on the Command and Control Workstation of the Future*, Report NPS52-88-027, Naval Postgraduate School, Monterey, California, 1988.
- Breden, W., and Zanolli, J., *Visualization of High-Resolution Digital Terrain*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1989.
- Weeks G. and Phillips E., *The Command and Control Workstation of the Future, Subsurface and Periscope Views*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1989.
- Defense Mapping Agency, "Digitizing the Future", DMA Combat Support Center, Washington, D.C., 1988.
- Boncal R.J., *A Study of Model Based Maneuvering Controls for Autonomous Underwater Vehicles*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1987.
- Zyda M., McGhee R., Kwak S., Nordman D., Rogers R, Marco., "Three-Dimension Visualization of Mission Planning and Control for the NPS Autonomous Underwater Vehicle", IEEE Journal of Oceanic Engineering, Vol. 15, No. 3, July 1990, pp 217-221.
- Zyda M., Pratt D., "Inexpensive 3D Visual Simulation as Workstation Exhaustion", Ausgraph 90 Proceedings, September 1990, pp. 313-325.

Smith, N.S., Crane, J. W., and Summey, D.C., "SDV Simulator Hydrodynamic Coefficients," NCSC Technical Memorandum 231-78, June 1978.

Gertler, M., and Hagen, G. R., "Standard Equations of Motion for Submarine Simulation," June 1967.

MacPherson, D. L., *A computer simulation study of mission planning and control for the NPS autonomous underwater vehicle*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1988.

Nordman D. B., *A computer simulation study of mission planning and control for the NPS autonomous underwater vehicle*, M. S. Thesis, Naval Postgraduate School, Monterey, California, June 1989.

Ong W., *A mission-planning expert system of the NPS autonomous underwater vehicle*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1990.

Ackley, Kurt., "Superworkstation, The Silicon Graphics 4D/240GTX Superworkstation", IEEE Computer Graphics and Applications, Vol. 9, No. 4, July 1989, pp. 71-83.

Wilhelms, Jane. "Dynamics for Everyone", IEEE Computer Graphics and Applications, Vol 7, No. 6, June 1987, pp 1-26.

Spiegel, M. R., *Applied Differential Equations, third edition*, Prentice-Hall, Inc., Englewood Cliffs, N. J. 1988, pp. 415-421.

Munson, S. A., *Integrated Support for Manipulation and Display of 3D Objects for the Command and Control Workstation of the Future*, M. S. Thesis, Naval Postgraduate School, Monterey, California, June 1989.

King, D., Prevatt R., *Rapid Prototyping of a Graphical User Interface*, M. S. Thesis, Naval Postgraduate School, Monterey, California, December 1990.

Barzel R., Barr, A., "Controlling Rigid Bodies with Dynamic Constraints", ACM SIGGRAPH 88 Course Notes, pp. E1- E26.

Silicon Graphics, Inc., "Graphics Library User's Guide, Version 2.0", Mountain View, California, 1988.

Sturman, D., "A Discussion of the Development of Motion Control Systems", ACM SIGGRAPH 1987 Course Notes, July 1987, pp.3-16.

Shoemake, K., "Animating Rotation with Quaternion Curves", IEEE Computer Graphics and Applications", Vol 19, No. 3, 1985, pp. 245-253.

Isaacs P., and Cohen M., "Controlling Dynamic Simulation with Kinematic Constraints, Behavior Functions, and Inverse Dynamics", Computer Graphics, Vol. 21, No. 4, July 1987, pp. 215-224.

Anderson, W., "*NPSME - An Interactive Tool for Material Characteristics Specification*",
M. S. Thesis, Naval Postgraduate School, Monterey, California, December 1990.

Fu, K., Gonzalez, R., Lee C., *Robotics: Control, Sensing, Vision, and Intelligence*,
McGraw-Hill, Inc., 1987.

INITIAL DISTRIBUTION LIST

- | | | |
|----|--|---|
| 1. | Defense Technical Information Center
Cameron Station
Alexandria, VA 22304-6145 | 2 |
| 2. | Library, Code 52
Naval Postgraduate School
Monterey, CA 93943-5002 | 2 |
| 3. | Dr. Michael J. Zyda
Naval Postgraduate School
Code CS, Department of Computer Science
Monterey, CA 93943-5100 | 7 |
| 4. | Dr. A. Healy
Naval Postgraduate School
Code ME, Department of Mechanical Engineering
Monterey, CA 93943-5100 | 1 |
| 5. | Commander Tom Jurewicz
341 Barbara Drive
Point Pleasant, New Jersey 08742 | 1 |
| 6. | Dr. Robert McGhee
Naval Postgraduate School
Code CS, Department of Computer Science
Monterey, CA 93943-5100 | 1 |
| 7. | David Pratt
Naval Postgraduate School
Code CS, Department of Computer Science
Monterey, CA 93943-5100 | 1 |

**END
FILMED**

DATE:

3-92

DTIC