TP 13676E

ICE NAVIGATION SIMULATION – PHASE II

Prepared for
Transportation Development Centre
Transport Canada

by
PhiloSoft Inc.
3608 Boulevard St. Charles, Suite #27c
Kirkland, Quebec
H9H 3C3

October 2000

TP 13676E

ICE NAVIGATION SIMULATION – PHASE II

by
Charles G. Marton
PhiloSoft Inc.

October 2000

Un sommaire français se trouve avant la table des matières.

Transport Canada   Transports Canada

PUBLICATION DATA FORM

1. Transport Canada Publication No. TP 13676E	2. Project No. 9748-9760-9761	3. Recipient's Catalogue No.
4. Title and Subtitle Ice Navigation Simulation – Phase II		5. Publication Date October 2000
		6. Performing Organization Document No.
7. Author(s) Charles G. Marton		8. Transport Canada File No. ZCD2450-368
9. Performing Organization Name and Address Philosoft Inc. 3608 St. Charles Blvd, Suite 27C Kirkland, Quebec Canada H9H 3C3		10. PWGSC File No. XSD-9-01484
		11. PWGSC or Transport Canada Contract No. T8200-9-9551/001/XSD
12. Sponsoring Agency Name and Address Transportation Development Centre (TDC) 800 René Lévesque Blvd. West Suite 600 Montreal, Quebec H3B 1X9		13. Type of Publication and Period Covered Final
		14. Project Officer C. Gautier

15. Supplementary Notes (Funding programs, titles of related publications, etc.)

Co-sponsored by Marine Safety Northern, and the Program of Energy Research and Development (PERD)

16. Abstract

Navigating ships in ice-infested waters has always been a risky and dangerous task. The requirement to develop an ice navigation simulator stems from a study conducted for the Transportation Development Centre in which the requirements for a training course were identified.

In Phase II the feasibility of semi-automating the analysis system and the transit model using the AKAC model were demonstrated. The Visual Generation Module and the display system were reworked to support the OpenFlight Database format. The main simulator platform was developed in conjunction with other modules, including the training aids module and the shipboard radar simulation module.

The main simulation software provides several key elements:

- Synthetic Aperture Radar (SAR) data conversion tools
- SAR data analysis tools
- Virtual out of window bridge view
- Database management system
- GPS and GYRO simulation
- Instructor and student functionality
- Bulletin Board Service data downlink
- Scaleable network operation

To date all of the changes in the modules that make up the simulator have been successfully integrated. The outcome of the project was positive and well-received by the review committee, which included representatives from the industry and the Canadian Coast Guard.

17. Key Words Marine, radar, simulation, ice navigation	18. Distribution Statement Limited number of copies available from the Transportation Development Centre

19. Security Classification (of this publication) Unclassified	20. Security Classification (of this page) Unclassified	21. Declassification (date) —	22. No. of Pages xii, 24	23. Price Shipping/ Handling

Transports Canada Transport Canada

FORMULE DE DONNÉES POUR PUBLICATION

1. N° de la publication de Transports Canada TP 13676E	2. N° de l'étude 9748-9760-9761	3. N° de catalogue du destinataire
4. Titre et sous-titre Ice Navigation Simulation – Phase II		5. Date de la publication Octobre 2000 6. N° de document de l'organisme exécutant
7. Auteur(s) Charles G. Marton		8. N° de dossier - Transports Canada ZCD2450-368
9. Nom et adresse de l'organisme exécutant Philosoft Inc. 3608, boul. Saint-Charles, bureau 27C Kirkland, Québec Canada H9H 3C3		10. N° de dossier - TPSGC XSD-9-01484 11. N° de contrat - TPSGC ou Transports Canada T8200-9-9551/001/XSD
12. Nom et adresse de l'organisme parrain Centre de développement des transports (CDT) 800, boul. René-Lévesque Ouest Bureau 600 Montréal (Québec) H3B 1X9		13. Genre de publication et période visée Final 14. Agent de projet C. Gautier

15. Remarques additionnelles (programmes de financement, titres de publications connexes, etc.)

Coparrainé par la Direction générale de la sécurité maritime, région du Nord, et le Programme de recherche et développement énergétiques (PRDE)

16. Résumé

Naviguer en eaux couvertes de glaces est depuis toujours une tâche risquée et dangereuse. La nécessité de développer un simulateur de navigation dans les glaces découle d'une étude menée par le Centre de développement des transports, qui a déterminé les critères que devrait respecter un cours de formation à la navigation dans les glaces.

La phase II du projet a démontré la faisabilité d'automatiser partiellement le système d'analyse; elle a également démontré la modélisation du déplacement des navires dans les glaces à l'aide du modèle développé par AKAC. Le module de visualisation et le système d'affichage ont été modifiés de façon à être compatibles avec le format OpenFlight Database. La plate-forme principale de simulation a été mise au point concurremment avec d'autres modules, y compris les modules d'aide à la formation et de simulation du radar embarqué.

Le logiciel principal de simulation comporte plusieurs éléments clés :

- outils de conversion des données recueillies par le radar à synthèse d'ouverture (RSO),
- outils d'analyse des données RSO,
- image virtuelle du pont du navire vu de la passerelle,
- système de gestion de la base de données,
- simulation du GPS et du gyrocompas,
- fonctionnalité instructeur et stagiaire,
- liaison avec un babillard électronique,
- adaptabilité à l'exploitation en réseau.

Jusqu'à maintenant, les chercheurs ont réussi à intégrer tous les changements des différents modules constitutifs du simulateur. Le projet a donné des résultats positifs, qui ont été bien accueillis par le comité d'examen, composé entre autres de représentants de l'industrie et de la Garde côtière canadienne.

17. Mots clés Marine, radar, simulation, navigation dans les glaces	18. Diffusion Le Centre de développement des transports dispose d'un nombre limité d'exemplaires.

19. Classification de sécurité (de cette publication) Non classifiée	20. Classification de sécurité (de cette page) Non classifiée	21. Déclassification (date) —	22. Nombre de pages xii, 24	23. Prix Port et manutention

CDT/TDC 79-005
Rev. 96

Canada

ACKNOWLEDGEMENTS

A number of individuals contributed to the completion of this phase of the project. I would like to extend extra special thanks to Charles Gautier of the Transportation Development Centre for his many valuable suggestions and thorough review of this and related documentation.

EXECUTIVE SUMMARY

This project was a result of a study conducted on behalf of the Canadian Coast Guard (TP 12496) that outlined the essential elements required to make up an International Ice Navigator Course. A significant component of this course involved training ice navigators with the aid of a simulator. The need for a simulator has been recognized by such countries as Finland, Russia, Sweden, Germany and Norway.

Different approaches have been considered for the implementation of this capability, hereafter called the Ice Navigation Simulator, and a number of facilities exist where large modifiable simulation engines could be used to perform this task. However, these systems use proprietary technologies and are not considered “open” systems. Besides being very expensive, they do not allow for widespread distribution and availability. Given state-of-the-art PC technology, complete with near workstation performance at a fraction of the cost, together with the explosion of multimedia and available virtual reality equipment, it is now possible to consider alternatives to the fixed, high-cost and proprietary systems currently in use for shipboard simulation. Therefore, a new approach to a realistic, low-cost implementation of the simulator was adopted.

The simulator encompasses the following elements:

- Simulation of ice on shipboard radar
- Simulation and management of remotely sensed data
- Simulation of visual aspect of ice, in daylight and night transit conditions
- Ship transit simulation (basic at this stage)
- Ice recognition and ice climatology training aids
- Ice regime entry rules training aid
- Ice Navigation systems and Electronic Charting and Display Information System (ECDIS) support

The Main Simulator Platform (MSP) ties these elements together to create an environment that visually and operationally resembles the ice navigation environment. Following the successful integration of the above items (Phase I), Phase II of the project included the development of and improvement in the following areas:

- Semi-automating the scenario generation tools to improve both the quality and level of analysis.
- Improving the visual system by switching to Open Flight Database Format to open the system for enhancements and reduce the requirement for expensive graphics hardware.
- Adding the transit simulation module to provide a more realistic motion model.

Although Phase II was a success there are still improvements to the system that can and should be done to raise the simulator to a world-class tool. These include:

- Providing better feedback to the operator.
- Optimizing the world sizes.
- Incorporating more training features such as the ice numerology system.

SOMMAIRE

Le projet est issu d’une étude menée pour la Garde côtière canadienne (projet TP12496), et qui établissait les principaux éléments d’un Cours international de navigation dans les glaces. Ce cours comprenait une composante importante : former les navigateurs dans les glaces à l’aide d’un simulateur. L’utilisation du simulateur pour assurer la formation a été reconnue par des pays comme la Finlande, la Russie, la Suède, l’Allemagne et la Norvège.

Différentes approches ont été étudiées pour la réalisation d’un simulateur de navigation dans les glaces. Il existe un bon nombre de systèmes qui pourraient satisfaire ce besoin avec de puissants moteurs de simulation. Or, ces systèmes utilisent des technologies brevetées et ils ne sont pas considérés «ouverts». En plus d’être très coûteux, ils ne bénéficient ni d’une vaste distribution ni d’une grande disponibilité. Avec la technologie avancée des PC, qui sont entièrement équipés et présentent à une fraction du coût une performance proche de celle d’un poste de travail spécialisé, et compte tenu de l’explosion du multimédia et de l’équipement de réalité virtuelle offert sur le marché, on dispose maintenant de solutions de remplacement efficaces des systèmes fixes brevetés, très chers, actuellement employés pour la simulation des navires. Aussi, une nouvelle approche a été adoptée en vue de satisfaire au besoin d’une installation de simulation à faible coût, produisant le degré de réalisme requis.

Le simulateur présente les caractéristiques suivantes :

- simulation des glaces sur le radar embarqué,
- simulation et gestion des données acquises par télédétection,
- visualisation de l’aspect de la glace, en conditions de navigation de jour et de nuit,
- simulation (encore élémentaire) du déplacement du navire,
- aides à la formation en reconnaissance et en climatologie des glaces,
- aide à la formation aux règles d’entrée en régime de glaces,
- systèmes de navigation dans les glaces et système de visualisation de cartes électroniques et d’information (ECDIS).

La plate-forme principale de simulation relie entre eux tous ces éléments pour créer un environnement qui reproduit visuellement, et du point de vue opérationnel, l’environnement de la navigation dans les glaces. Après l’intégration réussie des caractéristiques ci-haut (phase 1), le projet est entré dans la phase II, qui comprenait les activités ci-après de développement et de mise au point :

- semi-automatisation des outils de génération de scénario afin d’améliorer et la qualité et le niveau de l’analyse;
- amélioration du système de visualisation par le passage au format OpenFlight Database, qui permettra de rehausser les images et de réduire l’utilisation d’un matériel infographique dispendieux;
- ajout d’un module de simulation des déplacements du navire afin d’obtenir une modélisation plus réaliste du mouvement.

Si la phase II s’est révélée un succès, d’autres améliorations sont à prévoir pour faire du simulateur un outil de classe mondiale; entre autres :

- assurer une meilleure rétroaction pour l’opérateur,
- optimiser les dimensions du monde virtuel,
- incorporer d’autres outils de formation, par exemple le système de numérologie des glaces.

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

1.1. Scope

This document is a final report for the Ice Navigation Simulator – Phase II Project. The information described in this document defines the work performed during the execution of this project.

1.2. Overview

The Main Simulator Platform (MSP) was developed in order that a full and complete training platform could be implemented for ice navigation. The overall program objective was to develop a low-cost, PC-based Ice Navigation Simulation platform to train entry-level ice navigators. The MSP included the development of a scenario generation module (SGM); the base data management module (BDM), for synthetic aperture radar (SAR) and remotely sensed imagery; the visual simulation module (VSM); and the user interface to the system. The MSP project included the development of interfaces to auxiliary simulator modules such as the shipboard radar simulation module (SRSM) and the transit simulation module (TSM), as well as interfaces to the Ice Navigation system and electronic charting and display information system (ECDIS). In addition, the interfaces to various other training aid modules (TAMs) were implemented during the course of this project. These included the Ice Recognition and the Ice Regime Entry Rules Training Aids.

1.3. Document Overview

This document contains a brief description of the changes and additions to the main simulator software elements since the beginning of Phase II. All system software design documentation will be submitted at the same time as this final report. The key tasks described are listed below.

1) Purchase of the OpenGVS Software developer's kit (purchased Dec. 8, 1999) and graphics cards
2) Transit Simulation Module (TSM) implementation:
    - Main Simulator Platform (MSP) upgraded to accommodate new TSM
    - TSM testing and integration into existing Visual Simulation Module (VSM).
3) Scenario Generation Module (SGM) modifications included the addition of:
    - Freehand feature outline drawing
    - Line collision detection and vectorization
    - Edge detection in SGM
    - Additional analysis (10-20 times the number of features)
4) Recoding of Visual Generation Module (VGM) to support generation of OpenFlight 3D-worlds.
5) Recoding of VSM to support OpenGVS library and OpenFlight standard

2. MSP General Architecture

The MSP provides the data to and interfaces with various subsystems. The MSP software is composed of several components and dynamic link libraries. Figure 2-1 illustrates the relationship of the various software components that make up the Ice Navigation Simulator.

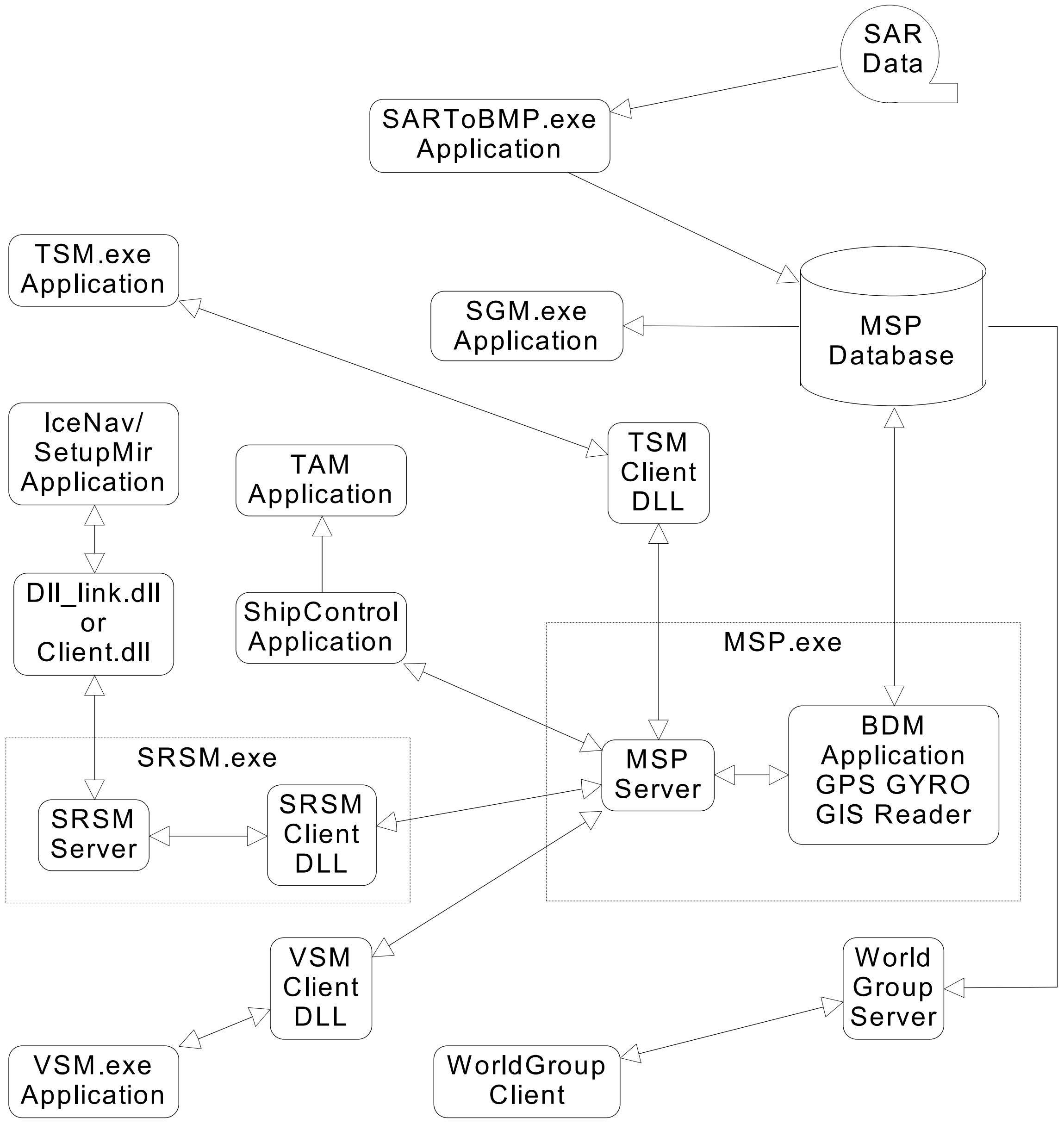


Figure 2-1 Simulator Applications and DLLs

3. TSM Implementation

The TSM algorithm was extracted and coded according to the information contained in the *Ice Manoeuvring Model Review* (12 March 1999).

3.1. Transit Model Parameters

Based on the requirement analysis study, it was determined that the following parameters were necessary to support the model.

Ship Parameters

1. Displacement:	In tonnes
2. LWL(L):	Load waterline length (m)
3. C_h:	1 for Inerta coating and 1.33 for bare steel
4. hull:	Rounded? Chined?
5. B:	Ship Beam (m)
6. T:	Ship Draft (m)
7. ψ:	Bow flare angle averaged over the beam
8. Φ:	Bow buttock angle averaged over the beam
9. Psmax:	Maximum design shaft power
10. D/LWL:	In Open Water (D: Turning Diameter at largest rudder angle)
11. PMB:	Percent parallel middle body.
12. Rudder position:	Fully Effective Rudders --- placed immediately behind the propellers Partially Effective Rudders --- not placed directly behind the propellers
13. Reamers:	If the ship has reamers then r = reamer width/paraller side (%)
14. Pitch System	Ducted Controllable Pitch System Open Controllable Pitch System Open Fixed Pitch System

Ice, Water and Weather Parameters

1. H:	Level ice thickness (m)
2. ρ:	Flexural strength of ice (KPa)
3. t:	Ice surface or air temperature in Celsius
4. Cs:	1.0 for saline, 0.85 for brackish, 0.75 for fresh water conditions

3.2. Derived Formulae

The following formulae were extracted from the report. (**Note:** Unless stated otherwise all units are metric.)

3.2.1. AKAC Model - Calculate Ice and Open Water Resistance

The AKAC resistance model was chosen for the simulator and formulae were derived for both rounded and chined hull types. They are described below.

$$R(v)_t = R(v)_{ow} + R(1m/s)_{ice} + R(>1m/s)_{ice} ;$$

Round hull:

$$R(v)_{ow} = (\text{Displacement})^{1.1} * (0.025*Fn + 8.8*Fn^5) \qquad \text{kN}$$

$$R(1m/s)_{ice} = 0.015*C_s*C_h*B^{0.7}*L^{0.2}*T^{0.1}*H^{1.5}$$
$$*(1- 0.0083*(t+30))*(0.63+0.00074* \ )$$
$$*(1+0.0018*(90-\Psi)^{1.6})*(1+0.003*(\Phi-5)^{1.5}) \qquad \text{MN}$$

$$R(>1m/s) = 0.009*v_{increase}/(g*L)^{0.5}*B^{1.5}*T^{0.5}*H_{ice}$$
$$*(1+0.0018*(90-\Psi)^{1.6})*(1+0.003*(\Phi-5)^{1.5})$$
$$*(1- 0.0083*(t+30))*C_h \qquad \text{MN}$$

Chined hull:

$$R(v)ow = 1224 * (\text{Displacement})^{0.25} * Fn^2 \qquad \text{kN}$$

$$R(1m/s)ice = 0.08+0.017*C_s*C_h*B^{0.7}*L^{0.2}*T^{0.1}*H^{1.25}$$
$$*(1- 0.0083*(t+30))*(0.63 + 0.00074 * \rho)$$
$$*(1+ 0.0018*(90-\Psi)^{1.4})*(1+0.004*(\Phi-5)^{1.5}) \qquad \text{MN}$$

$$R(>1m/s) = 0.009*v_{increase}/(g*L)^{0.5}*B^{1.5}*T^{0.5}*H_{ice}$$
$$*(1+0.0018*(90-\Psi)^{1.4})*(1+0.003*(\Phi-5)^{1.5})$$
$$*(1- 0.0083*(t+30))*C_h \qquad \text{MN}$$

where,

Displacement is in tonnes,

$Fn = v/(g*L)^{0.5}$

v:	in m/s
L:	Load waterline length (m)
G:	9.81 m/s^2
C_h:	1 for Inerta coating and 1.33 for bare steel
B:	Ship Beam (m)
T:	Ship Draft (m)
Ψ:	Bow flare angle averaged over the beam
Φ:	Bow buttock angle averaged over the beam
H:	Level ice thickness (m)
ρ:	Flexural strength of ice (kPa)
t:	Ice surface or air temperature in Celsius
Cs:	1.0 for saline, 0.85 for brackish, 0.75 for fresh water conditions

3.2.2. Net Thrust

Three models are used to calculate net thrust. The ship's propeller type determines which formula to use.

The parameters used to calculate thrust are listed below.

P_{smax}:	Maximum design shaft power in MW
x:	Decimal fraction of P_{smax} used
v:	Vessel speed in m/s
PP:	Propulsive performance in N/W

Ducted Controllable Pitch System:

T = P_{smax} * x * (1- 0.05*v/8)*PP **(MN)**
PP=[(0.15- 0.0082*v)+1.36*(0.95-x)*(0.051- 0.0028*v)] **(N/W)**
x=(x>0.95)? 0.95:x;

Open Controllable Pitch System:

T = P_{smax} * x * (1- 0.25*v/8)*PP **(MN)**
PP=[(0.116- 0.004*v)+1.5*(0.75-x)*(0.037- 0.0013*v)] **(N/W)**
x=(x>0.75)? 0.75:x;

Open Fixed Pitch System:

T = P_{smax} * x * (1- 0.25*v/8)*PP **(MN)**
PP=[(0.122- 0.0057*v)+1.5*(0.75-x)*(0.054- 0.0042*v)] **(N/W)**
x=(x>0.75)? 0.75:x;

3.2.3. Turning Diameter

In cases where the rudder(s) is positioned immediately behind the propeller(s), the steering system is considered fully effective. However, when the rudder is not placed directly behind the propellers, we have a partially effective steering system.

Vessels with Fully Effective Rudders:

$$\mathbf{D/L_{\text{at } H_{ice}=0.6*H_{1m/s}} = -0.00005*(PMB)^3 + 0.0115*(PMB)^2 - 0.0618*PMB + 4}$$

Vessels with Partially Effective Rudders:

$$\mathbf{D/L_{\text{at } H_{ice}=0.6*H_{1m/s}} = -0.0003*(PMB)^3 + 0.0372*(PMB)^2 - 0.1544*PMB + 6}$$

Vessels with Reamers:

$$\mathbf{D/L_{\text{at } H_{ice}=0.6*H_{1m/s}} = 0.569*r^4 - 5.4904*r^3 + 21.498*r^2 - 44.317*r + 47.8}$$

where,

D:	Minimum turning diameter
L:	Load waterline length
$H_{1m/s}$:	Maximum ice thickness that can be broken while ship speed is at 1 m/s
H_{ice}:	Ice thickness
PMB:	Ppercent parallel middle body

The D/L value at any ice thickness can be determined as follows:

If <u>$H_{ice}/H_{1m/s} > 0.6$</u>, then

$$\mathbf{D/L_{\text{at } H_{ice}} = [D/L_{\text{at } H_{ice}=0.6*H_{1m/s}} - D/L_{\text{at } H_{ice}=0}] \\ *[1+(0.17+0.5*(50 - PMB)/30)*(H_{ice}/H_{1m/s} - 0.6)*10] \\ + [D/L_{\text{at } H_{ice}=0}]}$$

Where, if **PMB>50, PMB=50**; if **PMB<20, PMB=20.**

If <u>$H_{ice}/H_{1m/s} < 0.6$</u>, then

$$\mathbf{D/L_{\text{at } H_{ice}} = [D/L_{\text{at } H_{ice}=0.6*H_{1m/s}} - D/L_{\text{at } H_{ice}=0}] \\ *[(H_{ice}/H_{1m/s})/0.6 - ((50 - PMB)/300)* \\ (-57.123*(H_{ice}/H_{1m/s})^3 + 14.246*(H_{ice}/H_{1m/s})^2 + 12.135*H_{ice}/H_{1m/s})] \\ + [D/L_{\text{at } H_{ice}=0}]}$$

Where, if **PMB>50, PMB=50**; if **PMB<20, PMB=20.**

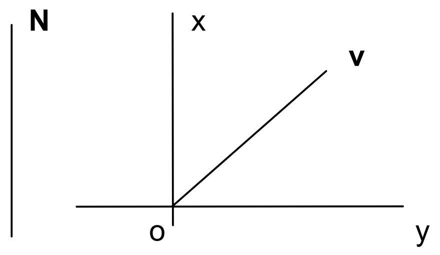
3.2.4. Turning Resistance

The factor for increasing the resistance is then:

$$\mathbf{-10.652*(L/R)^4 + 21.635*(L/R)^3 - 16.755*(L/R)^2 + 8.3223*(L/R)}$$

3.2.5. Ship Motion Dynamics

The ship's motion dynamics are computed based on simple Newtonian laws and physics.



The formula uses sequentially time ordered samples.
For Time t_o we have: Velocity v_o, Position (x_o,y_o) and Acceleration a_o.
For current Time t we need to get: v, (x,y) and a.

For a small Δt ($\Delta t = t - t_o$):

$$\Delta x = x - x_o = v_o * \cos\theta * \Delta t + (1/2) * a_o * \cos\theta * (\Delta t)^2;$$
$$\Delta y = y - y_o = v_o * \sin\theta * \Delta t + (1/2) * a_o * \sin\theta * (\Delta t)^2;$$
$$\Delta v = v - v_o = a_o * \Delta t;$$
$$a = (T - R)/M;$$

Where:

M:	Ship displacement
T:	Thrust
R:	Resistance

To calculate the change in direction while turning, we use:

$$\Delta\theta = \theta - \theta_o = 2*(v/D)* \Delta t;$$

Where D is the diameter of the turning circle. The results are then substituted into the formulae above.

3.3. TSM Operation

The following sequence exemplifies the new operation of the TSM.

1) TSM Initialization:
VSM initialization, to get the world name, ship name, environment, ship position.
Transit Model initialization to get ship parameter.

2) TSM Runtime:
Call Transit Model runtime to get new ship position, speed, heading, etc.
Send message to Communication Thread.
Call VSM runtime.

3) On Communication Thread message processing:
   Call TSMSetData() to set new ship position, speed, heading to MSP Server.
   Call VSMGetData() to get the environment information of new position and ship control information.
   Send message to TSM.

4) TSM OnGetData (message processing):
   Update the environment information of TransModel.
   Update the ship control information of TransModel.

The following is an example and description of configuration files for the **TSM**:
TSM.cfg

```
#
# TSM CONFIGURATION FILE
MSP_SERVER= MRI1                          ; name of machine running the MSP Server
TSM_DLL      = F:/TSMDLL/DEBUG/TSMDLL.dll  ; path-filename to the TSM dll
SHIP_PATH    = F:\MXIONG\WINVSM           ; path to the ship parameter file(s)
#
#  Following is an ice table: Ice type=flexural strength of ice (kPa)
First Year=1000;
Nilas=800;
Young=600;
Grey=500;
Grey-White=700;
Thin First Year=1000;
Medium First Year=1000;
Thick First Year=1000;
Old=1500;
Second Year=2000;
Multi-Year=3000;
Ice of Land Origin=1500;
Unknown=2500;
```

NOTE: These values above are relative estimates and need to be varied.

MV_ARCTIC.prm

```
#
# Ship dynamic parameters of MV_ARCTIC
mass           = 38900000;             Displacement (kg)
length         = 211.9;                Load waterline length (M)
coat           = 1;                    1 for Inerta coating and 1.33 for bare steel
roundHull      = True;                 True for round hull or False for chined hull
beam           = 22.9;                 Ship Beam (m)
draft          = 11;                   Ship Draft (m)
flareAngle     = 54.8;                 Bow flare angle averaged over the beam (Deg.)
buttockAngle   = 20.5;                 Bow buttock angle averaged over the beam (Deg.)
power          = 13428000;             Maximum design shaft power (Watt)
turning_D_L    = 3;                    In open water, minimum turning diameter/ship length
pmb            = 50;                   Percent parallel middle body (%)
rwp            = 0;                    Reamer width / parallel side (%)
rudderType     = RUDDER_FULL;          See the following to select a value
pitchType      = PITCH_DUCT_CONTROL;   See the following to select a value
```

# **rudderType** can be one of following:
    # **RUDDER_FULL**:
    #   Fully Effective Rudders placed immediately behind the propellers.
    # **RUDDER_PARTIAL**:
    #   Partially Effective Rudders not placed directly behind the propellers
    # **RUDDER_REAMER**:
    #   With a reamer
# **pitchType** can be one of following:

# **PITCH_DUCT_CONTROL**:	Ducted Controllable Pitch System
# **PITCH_OPEN_CONTROL**:	Open Controllable Pitch System
# **PITCH_OPEN_FIXED**:	Open Fixed Pitch System

3.4. New Data Used in Data Exchange Between TSM and MSP

The WEATHERINFO structure was replaced by the ENVIRON_INFO structure to fit the Transit Model. At runtime the TSM receives the environment information from the MSP server via this data structure. The following new members were added to this data structure.

float	**temperature**	**-** Atmospheric temperature
float	**seaType**	**-** 1.0 for saline, 0.85 for brackish, 0.75 for fresh water conditions
int	**featureType**	**-** Ice feature type
float	**thickness**	**-** Ice thickness
char	**iceAttr]**	**-** Ice attribute
int	**snowCover**	**-** In cm

3.5. TransModel Class

The **ShipRunTime()** function of this class is used to get the ship velocity, displacement and heading for the simulator. This method is called periodically for every small time interval as defined in the file VSM.cfg.

Ice Attributes are attainable from the MSP server for each position in the “gaming” world. The Transit Model requires the flexural strength of ice so CArray **m_cIceTable** is used to translate the ice type to a corresponding flexural strength. This table is loaded during initialization from the TSM initial file (TSM.cfg.) The ship’s parameters are loaded into the TransModel Class from the ship parameter file (*.prm).

The public methods of the CTransModel class are listed below.

CTransModel() - Constructor of the CTransModel class

virtual ~CTransModel() – Destructor

void AddIceTable() - This function adds ice flexural strength values based on ice type to the m_cIceTable

void ShipControl() - This method sets the ship control parameters (i.e., Throttle and Rudder)

void ShipRunTime() - This function calculates the ship attributes (position, velocity, heading, etc.) after an elapsed time interval

void ShipAttrSet() - This function is used to initialize and set the ship Attributes.

void ShipParamInit() - This function is used to set ship parameters (displacement, load waterline length, beam, draft, etc.)

void UpdateEnviron() - This functions updates the environment (temperature, ice type, ice thickness, etc.)

4. VSM Models

4.1. Description

The new CVSM class was re-written with OpenGVS graphics API library. The new VSM supports all of the features of the existing VGM with the exception of the screen capture feature, which is currently unavailable in OpenGVS.

The new VSM uses the Infinite Sky Utilities of OpenGVS to have its light source correspond to the sun's illumination of the earth. The sun's direction depends on the position of the earth, time of year and time of day.

The multi-channel is supported by the new VSM. When the VSM initializes, it will detect the number of 3D graphic boards to determine how many channels will be used. Now it can support up to eight channels and this number can easily be expanded. Finally, the new VSM uses the new OpenFlight database, which is prepared by the new VGM.

4.2. VSM Initialization

VSMInit() gets its initial information from a vsm.cfg file. The following is an example of such a file.

vsm.cfg

```
# VSM configure file
SIMULATOR_NAME          = ICENAVSIMU        ;Simulator program name
VSM_STATION             = ARCTIC            ;Machine on which VSM runs
MSP_STATION             = MRI1              ;Machine on which MSP server runs
#
# Location of DLLs and configure files
VSMDLL                  = F:\Vsm\dll\vsmdll.dll ;File path of vsmdll.dll
SHIP_INFO_DIR           = F:\Test\testModel\   ;Directory of ship configure files
WORLD_INFO_DIR          = F:\Test\testModel\   ;Directory of world configure files
#
# Directories used by the VSM
MODEL_DIR               = F:\Test\testModel\            ;Directory of world model files
TEXTURES_DIR            = F:\Test\testModel\textures\ ;Directory of texture files
#
#Adjustable factors used by the VSM
VIEW_CENTER_H       =0.0        ;Horizontally shifting scale of view centre
VIEW_CENTER_V       =0.0        ;Vertically shifting scale of view centre
VIEW_ANGLE          =55         ;Angle of view in degree
NORMAL_CLIP_FAR     =100000.0  ;Distance of far clipping plane
NORMAL_CLIP_NEAR    =0.8        ;Distance of near clipping plane
EYE_SIDE_WIDTH      =15         ;Distance from centre to edge (view from port or starboard)
```

```
EYE_RISE_RATE        =0.1         ;Distance for every frame when Up or Down button used
ITERATION_RATE       = 13         ;The rate of update in ms
TEXT_ON              = TRUE       ;If the text string of ship information is displayed
```

The VSM loads the communication dynamic link library according to the value of the VSMDLL entry. It then communicates with the MSP server to get the initial information. This information includes the ship name, world name and environment information.

The system will load the configuration files associated with the ship name and world name. The following are examples of these two kinds of initialization files.

MV_ARTIC.inf

```
#
# Ship's configure file for VSM
SHIP_ENLARGE_X           = 1    ;Enlarge the ship vertically (x)
SHIP_ENLARGE_Y           = 1    ;Enlarge the ship horizontally (y)
SHIP_ENLARGE_Z           = 1    ;Enlarge the ship perpendicularly (z)
SHIP_HIGH                = 10.0 ;Height (y) of the ship
EYE_HIGH                 = 19.5 ;Height (y) of viewpoint
EYE_POSITION             = 88.6 ;Position (z) of viewpoint
EYE_TO_EDGE              = 0.0  ;Position (x) of viewpoint
SWAY_ANGLE_FREQUENCE     = 0.01 ;Angle frequency of the swaying imitation of the ship
SWAY_MAX_ANGLE           =1     ;Maximum angle of the swaying imitation of the ship
```

gulftestsm.inf

```
#
# World's configure file for VSM - Offset latitude and longitude
OFFSET_LONGITUDE        = -64.6578     ;Offset of longitude
OFFSET_LATITUDE         =  48.4068     ;Offset of latitude
```

After loading the ship and world information, the VSM initializes the 3D world by calling the function **GV_sys_init()** of OpenGVS. OpenGVS calls the default callback function **GV_user_init()** to determine the number of channels, loads the world and ship models, initializes the sky, fog and cloud models, and sets the positions of all objects.

4.3. VSM Runtime

At runtime the TSM calculates the ship position, ship velocity and ship heading. It then calls **TSMSetDate()** to send this information to the MSP server and calls **CVSM::RunTime()** to update the information for the VSM. In this runtime function, the VSM first sets the ship and viewpoint positions as well as the environment information. Then it sets the sun's direction. To minimize CPU computational overhead, the sun's direction is calculated and set only once per minute. **ShipSway()** will simulate the ship swaying. The swaying angle is a random angle between 0 to SWAY_MAX_ANGLE, which is loaded from ship's configuration file. The frequency of swaying is also loaded from this file. Sharp changes of the swaying angle are filtered in this function. After updating all information for the world, the VSM calls **GV_sys_proc()** of OpenGVS. OpenGVS then calls the default runtime callback function **GV_user_proc()** to draw the 3D world.

4.4. VSM User Interface

All of the GUI buttons from the Old WinVSM are retained for operation in the new version of the VSM. A member method **ProcessKey()** of CVSM is used to receive commands from the

CWinVSMDlg class. This function is also used to adjust the viewpoint position relative to the ship. In the new VSM, the rate of rising and falling, and the distance from port and starboard to centre can be adjusted in the vsm.cfg file.

4.5. VSM Mods - Sunlight Illumination Angle

The VSM was modified to have its light source correspond to the sun's illumination of the earth. The new formula computes the sun's azimuth for every given latitude, longitude, time of year and time of day. The computation is continuous and not discrete as in the past.

The public methods of the CSunlight class are listed below.

CSunlight()	**-** Constructor
virtual ~CSunlight()	**-** Destructor
ANGLE_POS SunAngle()	**-** Calculates the elevation and azimuth of sunlight
ANGLE_POS NearSunPos()-	Calculates the nearest point at earth to sun

4.6. CVSM Class

The public methods of the **CVSM** class are listed below.

CVSM()	**-** Constructor
virtual ~CVSM()	**-** Destructor
void RunTime()	**-** Called by CWinVSMDlg::OnTimer() on runtime
void ProcessKey()	**-** Called by CWinVSMDlg. Controls view angle, zoom factor, etc.
BOOL VSMInit()	**-** Initializes the VSM Model

5. MSP Modifications

The MSP was changed to accommodate the new TSM. As with the TSM, the WEATHERINFO structure was replaced by the ENVIRON_INFO structure and the following members were added to this structure:

float	**temperature**	**-** Atmospheric temperature
float	**seaType**	**-** 1.0 for saline, 0.85 for brackish, 0.75 for fresh water conditions
int	**featureType**	**-** Ice feature type
float	**thickness**	**-** Ice thickness
char	**iceAttr]**	**-** Ice attribute
int	**snowCover**	**-** In cm

nSnowCover, **fTemperature** and **fSeaType** were also added to the **SessionRecord** structure.

Three items were added to the **Weather Information** and **Start Up** dialogs: **Temperature**, **Water condition** and **Snow cover**. It is now possible to set this information for the new TSM in the MSP server.

6. SGM Modifications

6.1. Description

The modifications of the SGM make it easier to use. For example, the user can click the mouse to generate discrete polygon points or use freehand to define a feature boundary. The vectorizing is then automatically performed to reduce the vertices in the defined feature shape. If the user checks the Auto. Detect tool button or the same item in the popup menu, the edge detection will be processed after the vectorizing. Figure 6.1-1 demonstrates the process of adding a feature shape.

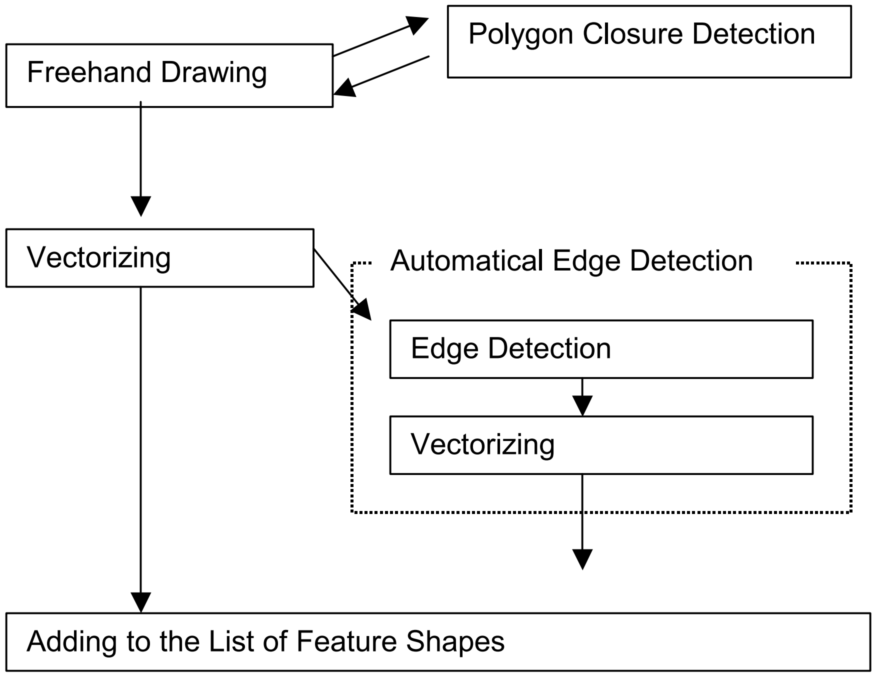
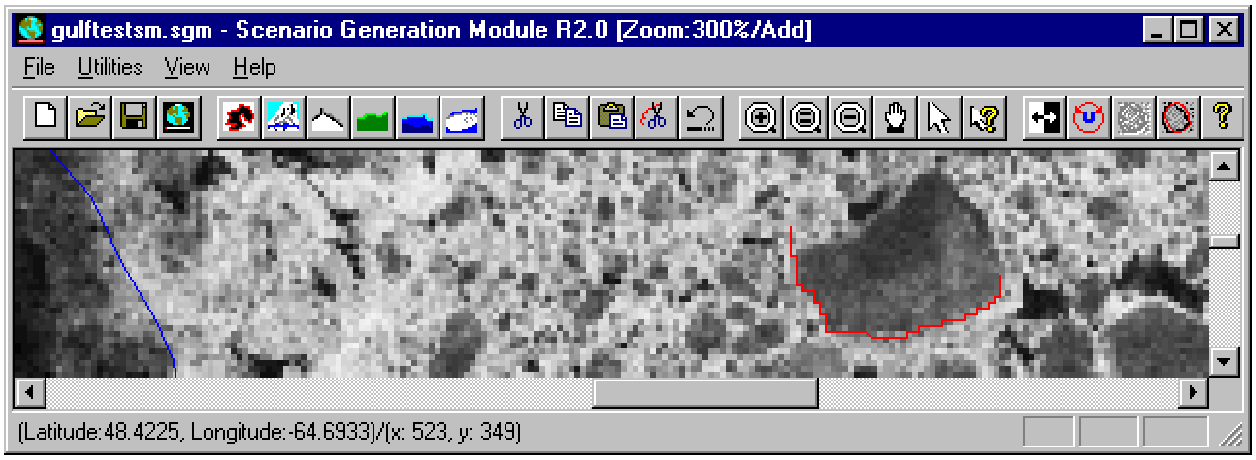

Polygon Closure Detection
Freehand Drawing
Vectorizing
Automatical Edge Detection
Edge Detection
Vectorizing
Adding to the List of Feature Shapes

Figure 6.1-1 Process of Adding a Feature Shape

6.2. Freehand Drawing

The CFreehand class is used by the SGM to reduce the number of mouse clicks when defining a feature boundary. Whereas previously the operator needed to click the mouse to identify the vertices of each bounding polygon, the operator now only needs to “draw” a closed freehand shape. The CVectorize class later vectorizes this shape. Figure 6.2-1 shows a feature boundary, which has been drawn freehand.


gulftestsm.sgm - Scenario Generation Module R2.0 [Zoom:300%/Add]
File
Utilities
View
Help

[Latitude:48.4225, Longitude:-64.6933]/[x: 523, y: 349]

Figure 6.2-1 Freehand Drawing

The class CArray **m_cFreehandPoints** is used to store the points that are drawn freehand. Another class, CArray **m_nSegment**, is used to store the segment information of points.

During the freehand drawing process it is possible that the leading and trailing segments will need to be discarded (e.g., drawing the closed shape corresponding to the Greek letter alpha). The methods **RemoveHeadBefore()** and **RemoveTailAfter()** provide this capability.

CFreehand has several member functions. The public interfaces for this class are listed below.

CFreehand();	- Class constructor
virtual ~CFreehand();	- Class Destructor

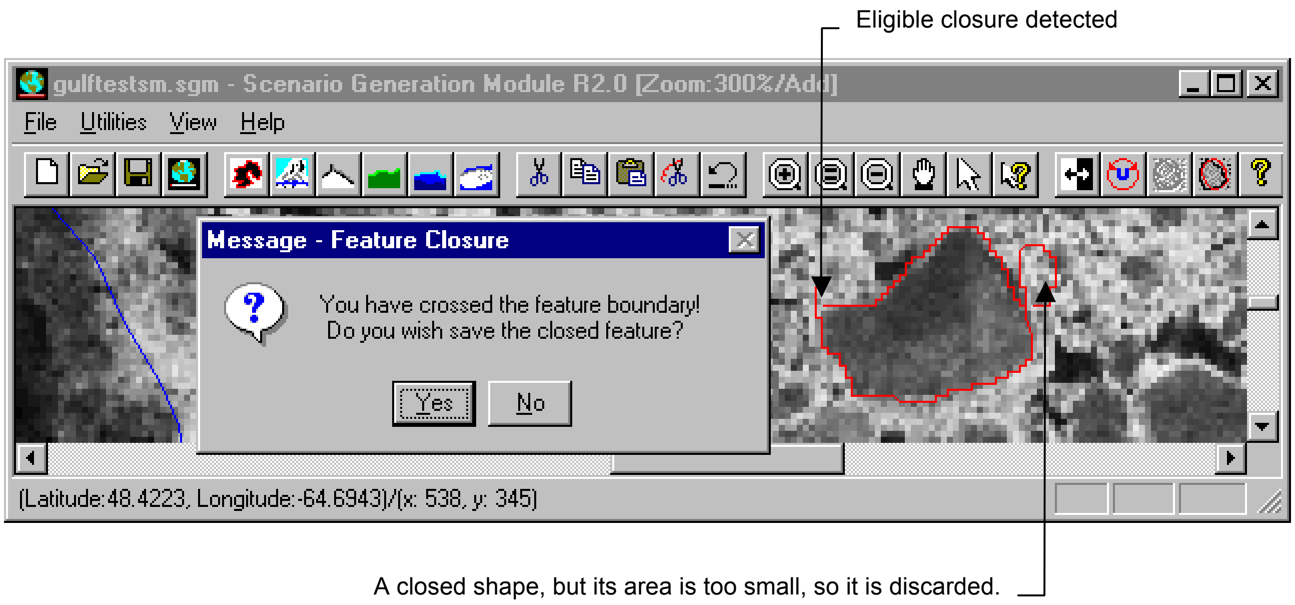
During the freehand drawing process it is possible that the leading and trailing segments will need to be discarded. This is partly because the start and end points do not meet but the shape was meant to be closed.

void RemoveHeadBefore()	- Removes the leading segment
void RemoveTailAfter()	- Removes the trailing segment
BOOL IsSegment()	- Determines if there are any points in m_cFreehandPoints
int CutLastSegment()	- Removes the last sSegment
CRect m_cPointRect	- Defines a minimum rectangle that can include all m_cFreehandPoints
CPoint * ExportPoints()	- Exports the pointer of m_cFreehandPoints
int GetPointsNumber()	- Returns the number of points in m_cFreehandPoints
void DrawAllPoints()	- Draws all freehand segments on screen
void DrawLast()	- Draws only the last segment on screen

void ResetPoints()	**-** resets the m_cFreehandPoints array to empty
void AddPoint()	**-** adds a point to m_cFreehandPoints. BSegment. Means this point will become a segment end point

6.3. Polygon Closure Detection

When freehand drawing a feature boundary, the program must automatically check if the drawn segments are closed. When a closed shape is found, the program compares the area of this polygon with the float variable **m_fNoiseScale**. If this area is less than **m_fNoiseScale** the shape is automatically discarded. Otherwise the Feature Closure dialog pops up to ask whether the user wants to save the closed feature. Figure 6.3-1 demonstrates this procedure.


Eligible closure detected
gulftestsm.sgm - Scenario Generation Module R2.0 [Zoom:300%/Add]
File Utilities View Help
Message - Feature Closure
Yes
No
A closed shape, but its area is too small, so it is discarded.

You have crossed the feature boundary!
Do you wish save the closed feature?

(Latitude:48.4223, Longitude:-64.6943)/(x: 538, y: 345)

Figure 6.3-1 Polygon Closure Detection

This class is also used to detect closure of a polygon when vertices are moved or deleted, or if a new shape is added. If a shape is not a polygon, the program cancels this action and informs the user.

The public methods of the **CPolyCloseDetect** class are listed below.

CPolyCloseDetect(float fNoise)	**-** Use this constructor function to set the minimum size of the polygon
virtual ~CPolyCloseDetect()	**-** Destructor
BOOL IsLegalShape()	**-** Checks if a shape is a polygon
BOOL IsInALine()	**-** Checks if a third point is in the segment that is defined by the first and second points
float CalcPolygonArea()	**-** Calculates the area of a polygon
int IsPointCrossLines()	**-** Checks the intersection of lines

CPoint m_cCrossPoint	**-** Stores the last crossed point
BOOL m_bIsNoise	**-** Signifies that the last detected polygon is too small

6.4. Vectorization

After using freehand to draw and close a feature boundary, it is necessary to vectorize the shape into a discrete set of points so that the triangulation algorithm in the VGM can compute the triangles necessary to generate the visual world. Figure 6.4-1 shows the change of the added shape after vectorizing.

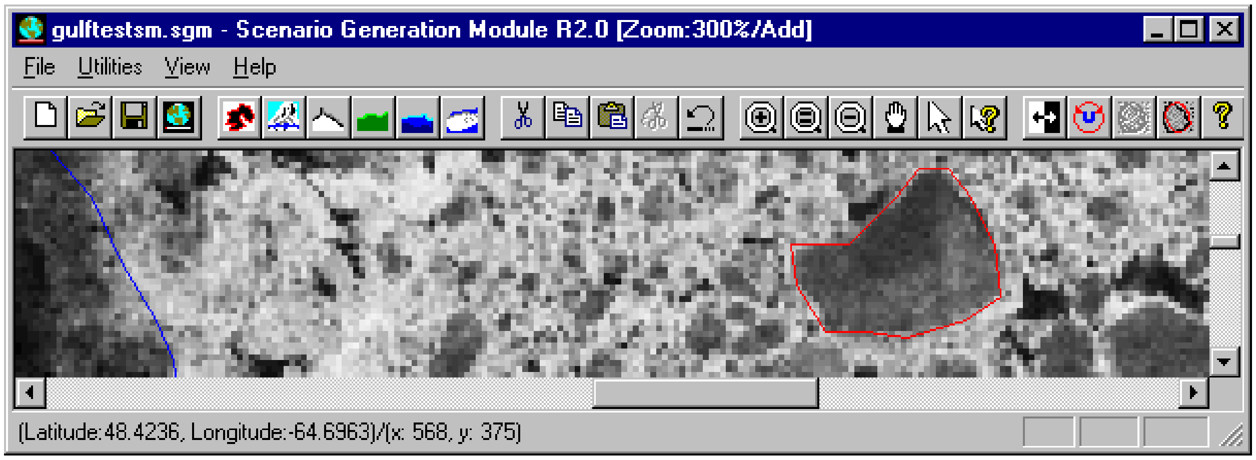

gulftestsm.sgm - Scenario Generation Module R2.0 [Zoom:300%/Add]
File
Utilities
View
Help
(Latitude:48.4236, Longitude:-64.6963)/(x: 568, y: 375)

Figure 6.4-1 Vectorization

To vectorize the feature boundary, it is necessary to discard some points from the freehand shape. To do this, the "importance" of every point is calculated. This importance is then compared to the importance of every other point and the less important points are discarded.

At least three points can determine the importance of the middle point. Importance means whether or not a point is to be kept as a vertex. A point is considered not important if:

a) the first point and middle point are very close;

b) the middle point and end point are very close;

c) the middle point is not far from the line connecting the first point and end point.

The CArray **m_cCVectorizePoints** is used to store the points while CVectorize vectorizes them. **m_nMaxVertices** is used to set the maximum number of points that can be left after vectorization and **m_nMinVertexDist** is used to set the minimum distance between two adjacent points.

The public methods of the CVectorize class are listed below.

CVectorize()	**-** Constructor of the CVectorize class. We can set the m_nMaxVertices and m_nMinVertexDist values thought this constructor. MAX_VERTICES (= 1000) is the default value for m_nMaxVertices. MIN_VERTEX_DIST (= 5) is the default value for m_nMinVertexDist.

virtual ~CVectorize() - Destructor of the CVectorize

void AddPoint() - Adds a point to m_cCVectorizePoints

void CVectorizeLoadPoints() - Adds groups of points to m_cCVectorizePoints

int FillShape() - Fills out a shape by using m_cCVectorizePoints, and reduces the number of points to a maximum number as defined in m_nMaxVertices

6.5. Edge Detection

6.5.1.User Interface

Edge Detect and Auto. Detect menu items were added to the SGM pop-up menu. These two items were also added in the toolbar of the SGM to be able to invoke these functions from a more visible interface as well. Figure 6.5-1 illustrates the new SGM interface.

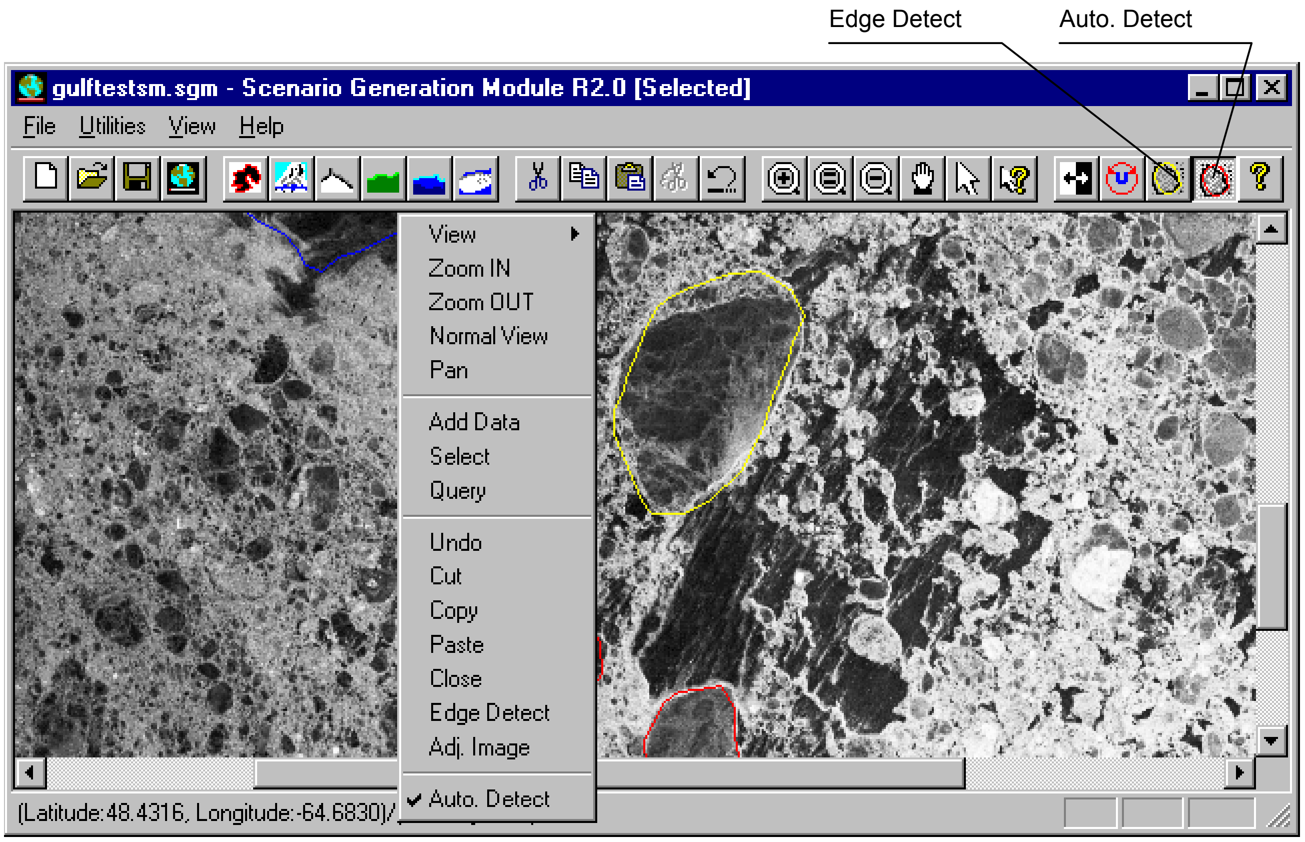


Figure 6.5-1 Edge Detection User Interface

When the user selects one or more feature shapes, the Edge Detect algorithm is enabled. The user can then select this menu item to process edge detection for selected feature shapes. Figure 6.5-2 shows the results of edge detection for the selected shape.

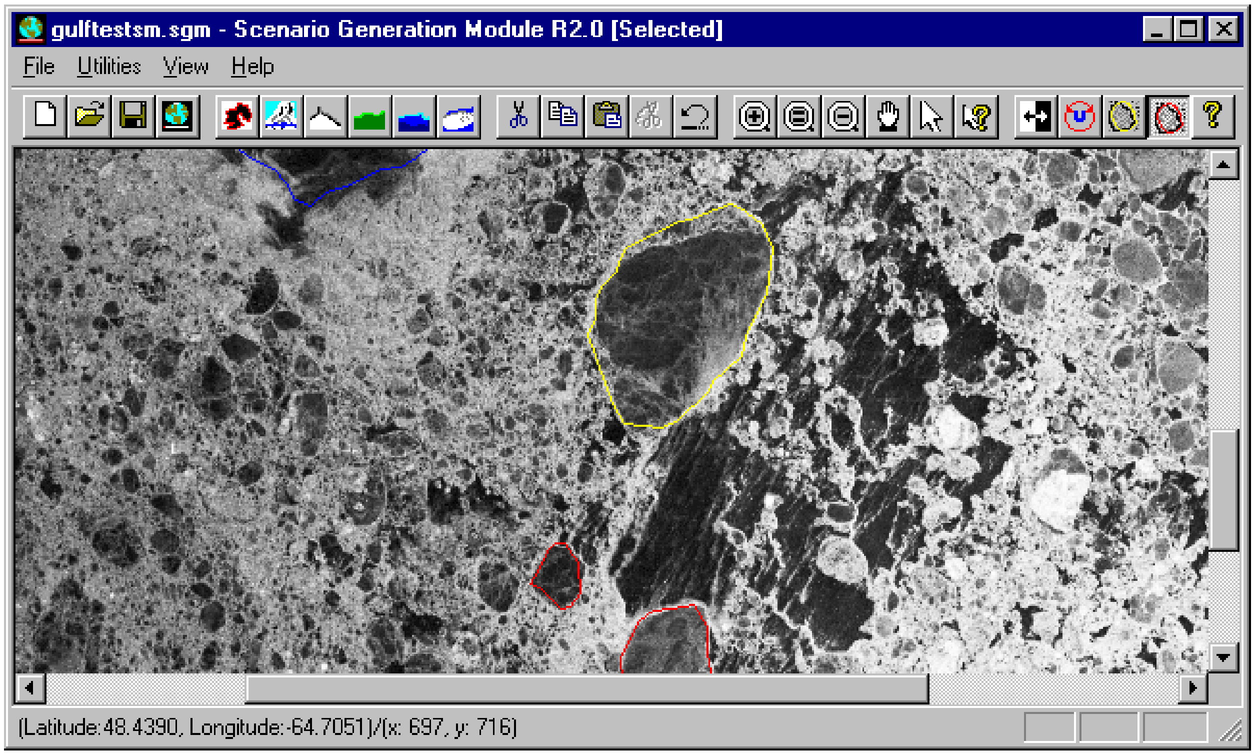

gulftestsm.sgm - Scenario Generation Module R2.0 [Selected]
File Utilities View Help
(Latitude:48.4390, Longitude:-64.7051)/(x: 697, y: 716)

Figure 6.5-2 Edge Detection

If the Auto. Detect item is checked by the user, the edge detection algorithm will be automatically applied while a feature shape is being added to the shape list.

6.5.2. Description for Edge Detection

After receiving an Edge Detect command, the program inputs the selected shape to the **CEdgeDetect** Class, and performs edge detection on the corresponding shape. If any edge is detected, the SGM uses the **CVectorizing** Class to vectorize the points that are exported from the **CEdgeDetect** Class. A new shape is composed to replace the old one. At the same time, this processing is added to the Undo list, so that the user can use the Undo button to cancel the edge detection processing.

The Auto. Detect function is only effective in the Add Mode. If it is checked, the edge detection is automatically performed on new feature shapes that are subsequently added.

The **CEdgeDetect** Class is designed to process edge detection. It loads points from a shape. All polygon points are first stored in the member variable **m_cPoints**. For every segment in this polygon, a trapezoid is calculated to prepare the searching line. The four composing segments of this trapezoid are two parallel segments and two dividing segments. Figure 6.5-3 demonstrates this trapezoid.

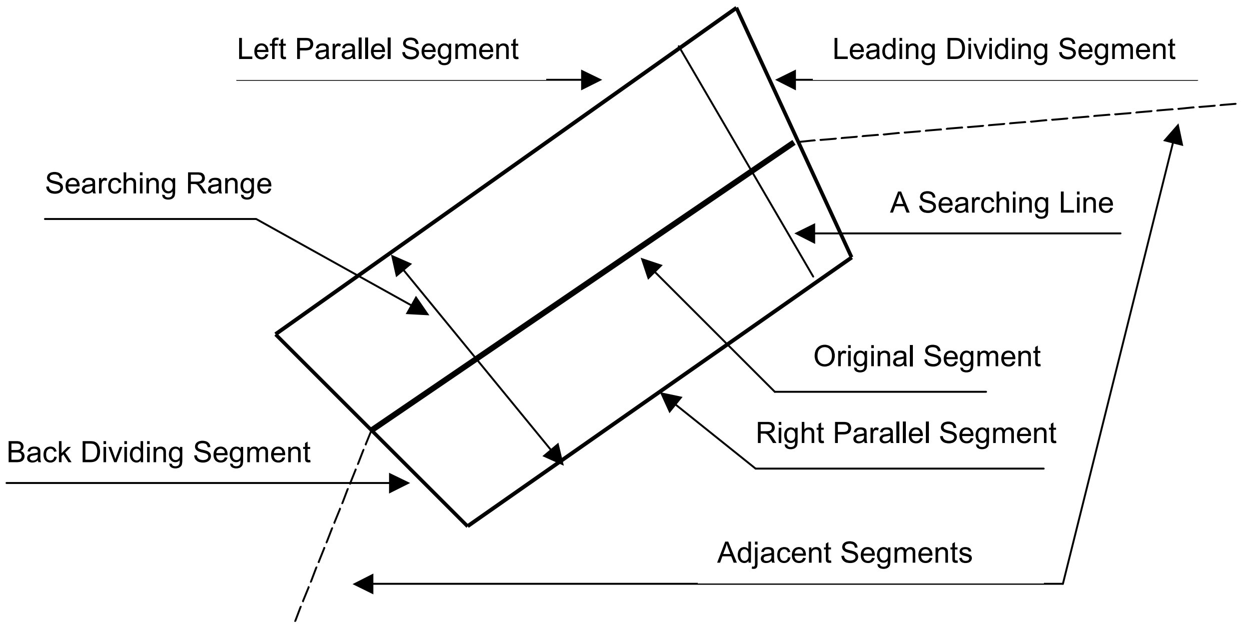


Figure 6.5-3 Searching Trapezoid

The searching range is the width of the searching band of this polygon. A float variable **EDT_RANGE_AREA_RATIO** is used to calculate this width. It is the ratio of the half width of the searching range and the square root of the area of this polygon. For every point in the original segment, a searching line is calculated according to its trapezoid. All points of a searching line are filled in a variable of **EdgePoint** structure. This variable is then added as an element into the **m_sEdgePoint** array. After the searching points are prepared as described above, edge detection is performed to find the edges in the search band. A simple algorithm is used to perform this process. If any edge is found, **CEdgeDetect** class selects the edges to compose the new boundary, smoothes the boundary and fills the points of this boundary to **m_cPoints**.

The index in the **EdgePoint** structure is used to address each point in the orthogonal line. The value of this index is ordered from the inside to the outside of the polygon. While edge detection is being performed, an average value is calculated for every point on the orthogonal line. The average value is computed based on the same index value of adjacent orthogonal lines. This calculated average is compared for each adjacent point along the drawn line. The point that is most different from the average value along an orthogonal line is marked. This is repeated for each point along the drawn polygon. If two adjacent "edge marker" points are too far apart, then these two points are not considered as being part of the same line. If there are enough marked points that are close enough together, then all the accumulated points become edge points.

6.5.3. CEdgeDetect Class

The public methods of the **CEdgeDetect** class are listed below.

CEdgeDetect()	**-** Constructor
virtual ~CEdgeDetect()	**-** Destructor
int GetPointsNumber()	**-** Gets the number of detected points

BOOL ProcessEdgeDetect()	**-** Detects the edges
CPoint * ExportPoints()	**-** Exports the detected points
void SetPointers()	**-** Sets the **CSGMDoc** and **CPolyCloseDetect** pointers to **CEdgeDetect**
BOOL SetPoints()	**-** Input points to **CEdgeDetect** class

6.6. Rubber Banding

This feature was added for ease of use even though it is not within the scope of this project.

When the mouse is used to drag a vertex in a feature shape, a rubber-banded line appears on the screen as the vertex is moved around the screen. This function is handled by **DrawRubberTrack()** in CSGMView class. In this function, old lines are erased before the new ones are drawn.

6.7. Dragging Features

This feature was added for ease of use and is not within the scope of this project. When the mouse is used to drag features, temporary outlines of the features appear on the screen as the features are moved around the screen. At the same time, a check is performed on the boundary to prevent the user from dragging features out of the world space. This function is handled by **DragFeatureMove()** in CSGMView class.

7. VGM

The new VGM was re-written to generate the 3D world in OpenFlight format and takes the ASCII file exported from the SGM as its input file. The SGM processes all information of the features, analyses them, divides them into triangles, puts textures on them and generates the OpenFlight format database for VSM.

7.1. VGM User Interface

Figure 7.1-1 shows the user interface of the VGM.

The Select Area button is used to select a world features description file, which is created by the Tool→Export ASCII Feature menu in SGM. The selected file name is shown in the edit box left of this button. At the same time, the same directory is displayed in the edit box left of the Texture Dir button. This is first step in generating a virtual world.

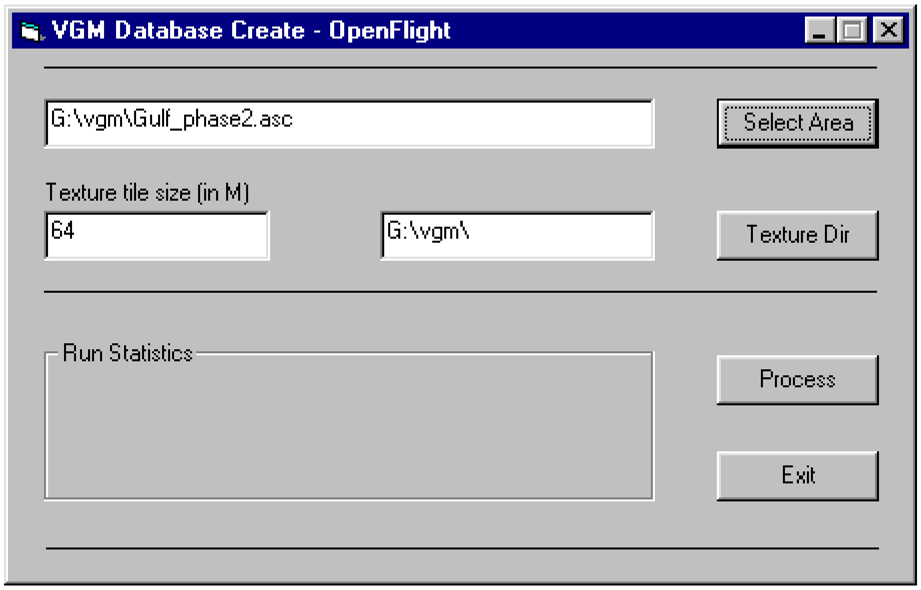
The Texture Dir button is used to select the directory of the textures of every type material. The selected directory is shown in the edit box left of this button. The default directory is the same as the world features description files.

The Texture Tile Size (in M) edit box is used to indicate a texture's real size in the world. Its unit of measure is expressed in meters. It is used to calculate the feature triangle point's u and v values. The default value is 64 (i.e., one texture image is 64 m by 64 m in the virtual world).

The Process button is used to start the process of generating the world model database.

The Exit button is used to exit the VGM. While generating the world model database, this button can be used to stop the generation.

The Run Statistics group box is used to show status information while generating the world model database. When generation is complete, the number of triangles in the virtual world will be shown here.



G:\vgm\Gulf_phase2.asc

Texture tile size (in M) 64

G:\vgm\

Figure 7.1-1 VGM User Interface

7.2. VGM Processing

After selecting a world feature description file, the user can click the Process button to generate the virtual world's OpenFlight database. The following sequence of steps occurs while generating the OpenFlight database. The program:

1. Opens the world description file and uses this file title to create a debug file whose extension is "dbg" and an OpenFlight database whose extension is "flt". For example, if the world description file is "Gulf.asc", the debug file name will be "Gulf.dbg" and the OpenFlight database name will be "Gulf.flt".
2. Initializes the OpenFlight database.
3. Inserts all textures into the OpenFlight database, and sets each texture's attribute.
4. Reads the world description file to get the world corners.
5. Scans through the rest of the world description file to calculate the total number of features in the world.
6. Performs the following steps for each feature in order:
    - Reads feature attributes from the world description file and saves the attributes.
    - Changes the feature polygon's points, originally described by longitude and latitude, to distance from the first world corner in meters.
    - Uses these points to create a ".poly" file used by "triangle.exe" to divide the polygon into triangles.
7. Sorts all features by their area in descending order.
8. Goes through the feature sorted queue for each feature and performs the following steps in order:
    - Checks if the current feature overlaps other features.
    - Opens the ".node" file that is created by "triangle.exe" to make each feature polygon node's vertices.
    - Calculates each feature polygon node's u and v depending on the Texture Tile Size value.
    - Sets the feature triangle's texture index depending on its type and age.
    - In the OpenFlight database, creates a group node for each feature ploygon and a object node for each triangle in the ploygon, then sets the attribute of object nodes.
9. Writes the OpenFlight database.
10. Deletes all of the temporary work files.

7.3. VGM Configuration

The VGM will create an OpenFlight database for the selected world features description file. The OpenFlight database is saved in the same directory as the description file. The title of the description file is used to name the OpenFlight database.

It is required that the model directory name be contained in the VSM configuration file. The OpenFlight database should also be copied to the model directory in order for the VSM to be able to load it.

Finally, the VSM configuration file should contain a texture directory name. This texture directory may be different from the texture directory that the VGM uses. If these directories are different, then the textures will need to be in both directories so that the VSM can find them.

8. Conclusion and Recommendations

The project was an overall success. Improvements, however, can still be made to the quality of the visual worlds, including the resolution and realism of the scenes. But the basic framework is functional and could be made field-operative and useful today. Furthermore, some optimization of the world sizes could and should be done to make the system more portable and available to a wider user group. The following improvements would elevate the simulator from strictly an R&D effort to a world-class simulation and training tool.

- Additional visual and scenario files. There are currently only two worlds. More variety is required.
- Incorporation of other ship types and classes. There is currently only one: the M.V. Arctic.
- Optimizing the world sizes. Currently the worlds are larger than they need to be and are causing less than optimal redraw rates. The ability to switch worlds without restarting the simulator is desirable.
- Incorporating more training features such as the ice numerology system, or a Red, Green and Yellow indicator system indicating dangerous conditions, hazards, warnings or to proceed with caution. Currently there is no indication that the operator is entering a dangerous area.
- Image resolution synthesis of lower radar range settings.
- Ship's track – allows you to see easily where you have been.
- Bridge Sound – provides engine control feedback.
- More ship databases to train operators in the same ice conditions but with different hull and propulsion strengths.
- Assessment and/or warning of potential damage to a ship if it hits ice (e.g., a bergy bit) at too high a speed.
- Better feedback to the operator such as vibration in the steering and throttle controls.
- Motion – adds the physical realism of being on the ship's deck.
- An integrated course syllabus that would lead the student through a set of standardized exercises.