# Hyper-Real-Time Ice Simulation and Modeling Using GPGPU

By
©Shadi Alawneh, B. Eng., M. Eng.

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree of

Doctor of Philosophy

Memorial University of Newfoundland

May 2014

Doctor of Philosophy (2014) Memorial University of Newfoundland
(Electrical and Computer Engineering) St. John's, Newfoundland

Title: Hyper-Real-Time Ice Simulation and Modeling Using GPGPU

Author: Shadi Alawneh, B. Eng. (Jordan University of Science and Technology), M. Eng. (Memorial University of Newfoundland)

Supervisor: Dr. Dennis Peters
Co-Supervisor: Dr. Claude Daley

# Abstract

Simulation of the behaviour of a ship operating in pack ice is a computationally intensive process to which General Purpose Computing on Graphical Processing Units (GPGPU) can be applied. GPGPU is the use of a GPU (graphics processing unit) to do general purpose scientific and engineering computing. The model for GPU computing is to use a CPU and GPU together in a heterogeneous co-processing computing platform. The sequential part of the application runs on the CPU and the computationally-intensive part is accelerated by the GPU. From the users perspective, the application just runs faster because it is using the high-performance of the GPU to boost performance. This thesis presents an efficient parallel implementation of such a simulator developed using the NVIDIA Compute Unified Device Architecture (CUDA). This simulator can be used to give users the ability to analyze ice-interactions for design, assessment and training purposes. This thesis also describes the execution of experiments to evaluate the performance of the simulator and to validate the numerical modeling of ship operations in pack ice. It also describes the useful applications that have been done using this simulator in planning ice management activities.

# Acknowledgements

I would like to express my sincere appreciation for the assistance and guidance of my supervisors, Dr. Dennis K. Peters and Dr. Claude Daley, in the preparation of this thesis and for their thoughtful guidance, constructive criticism and mentorship throughout my many years under their tutelage. Dr. Claude Daley collaborated with me on the GPU ice simulation equations in Chapter 3, Appendix A.1, and on the two applications of planning ice management activities in Chapter 6. [11, 10]

In addition I am idebted to my supervisory committee, Drs. Bruce Colbourne, Theodore Norvell, who have each taken the time to offer suggestions and guidance to help to improve the quality of this work. Dr. Bruce Colbourne collaborated with me on the autopilot algorithm for steering the vessel back on track in Chapter 3 and Appendix A.1.2.

Many friends and colleagues who have been associated with the Sustainable Technology for Polar Ships and Structures (STePS$^2$) Research Group, either as faculty, staff, students or visiting scholars, have offered support and suggestions that have undoubtedly contributed to this work. In particular, Roelf C. Dragt collaborated with me on the experiments to validate the generic GPGPU model functionality in

Chapter 6.

The financial support was received from: ABS, Atlantic Canada Opportunities Agency, BMT Fleet Technology, Husky Oil Operations Ltd, Research and Development Council, Newfoundland and Labrador and Samsung Heavy Industries.

Last but not least my father (Ghazi) and mother (Moyasser), It is just for being you.

# Publications

1. Claude Daley and **Shadi Alawneh** and Dennis Peters and Gary Blades and Bruce Colbourne. Simulation of Managed Sea Ice Loads on a Floating Offshore Platform using GPGPU-Event Mechanics. *Accepted to The International Conference and Exhibition on Performance of Ships and Structures in Ice (ICETECH 2014), July 2014, Banff, Alberta, Canada.*

2. Claude Daley and **Shadi Alawneh** and Dennis Peters and Bruce Colbourne. GPU-Event-Mechanics Evaluation of Ice Impact Load Statistics. *Proc. The Offshore Technology Conference (OTC 2014), February 2014, Houston, Texas, USA.*

3. **Shadi Alawneh**, Roelof Draget, Dennis Peters, Claude Daley and Stephen Bruneau. Hyper-Real-Time Ice Simulation And Modeling Using GPGPU. *Submitted to IEEE Transactions On Computers, November 2013.*

4. Steven Chaulk, **Shadi Alawneh**, Dennis Peters, Haochen Zhang, Claude Daley and Gary Blades. Ice Floe Simulator. Poster, Newfoundland Electrical and Computer Engineering Conference, November 2013, St. John's, NL, Canada.

5. **Shadi Alawneh** and Dennis Peters. 2D Triangulation of Polygons on CUDA. *Proc. The 2013 International Conference on High Performance Computing & Simulation (HPCS 2013) , July 2013, Helsinki, Finland. (Acceptance rate: 39.5%)*

6. **Shadi Alawneh**, Dennis Peters and Roelof Dragt. Ice Simulation Using GPGPU. Poster, The International GPU Technology Conference (GTC 2013), March 2013, San Jose, California.

7. Roelof Dragt, Stephen Bruneau and **Shadi Alawneh**. Design and Execution of Model Experiments to Validate Numerical Modelling of 2D Ship Operations in Pack Ice. In: *Proc. Newfoundland Electrical and Computer Engineering Conference, November 2012, St. John's, NL, Canada.*

8. Claude Daley, **Shadi Alawneh**, Dennis Peters, Bruce Quinton and Bruce Colbourne. GPU Modeling of Ship Operations in Pack Ice. *Proc. The International Conference and Exhibition on Performance of Ships and Structures in Ice (ICETECH 2012), September 2012, Banff, Alberta, Canada.*

9. **Shadi Alawneh** and Dennis Peters. Ice Simulation Using GPGPU. In: *Proc. 14th IEEE International Conference on High Performance Computing and Communications (HPCC-2012), June 2012, Liverpool, UK. (Acceptance rate: 26.2%)*

10. **Shadi Alawneh** and Dennis Peters. Enhancing Performance of Simulations using GPGPU. In: *Proc. Newfoundland Electrical and Computer Engineering*

*Conference, November 2011, St. John's, NL, Canada.*

11. Justin Adams, Justin Sheppard, **Shadi Alawneh** and D. Peters. Ice-Floe Simulation Viewer Tool. In: *Proc. Newfoundland Electrical and Computer Engineering Conference, November 2011, St. John's, NL, Canada.*

# Contents

# List of Figures

# List of Tables

# List of Acronyms

| Acronym | Description |
|---|---|
| GPGPU | General Purpose Computing on Graphical Processing Units |
| CUDA | Compute Unified Device Architecture |
| GPUs | Graphics Processing Units |
| STePS$^2$ | The Sustainable Technology for Polar Ships and Structures |
| IEMM | Ice Event Mechanics Modeling |
| GEM | GPU Event Mechanics |
| NSE | Navier Stokes Equations |
| PDEs | Partial Differential Equations |
| CSG | Constructive Solid Geometry |
| SCS | Sequenced Convex Subtraction |
| SMP | Symmetric Multiprocessor |
| SPMD | Single Program Multiple Data |
| MPMD | Multiple Program Multiple Data |
| MPI | Message Passing Model |

# Chapter 1

# Introduction

The Sustainable Technology for Polar Ships and Structures project (referred to as STePS$^2$)[1] supports sustainable development of polar regions by developing direct design tools for polar ships and offshore structures. Direct design improves on traditional design methods by calculating loads and responses against defined performance criteria. The deliverables of the project include a numerical model which accurately handles collision scenarios between ice and steel structures. The research described in this thesis is to use General Purpose GPU computing, or GPGPU, to implement some of the numerical models in this project.

Sea ice is a complex natural material that presents a challenge to ships and offshore structures. The idea described here allows the *practical* and *rapid* determination of ship-ice, ice-ice and ice-structure interaction forces and effects in a sophisticated ice regime. The term *rapid* is meant to mean at least real-time with the aim to be

[1] http://www.engr.mun.ca/steps2/index.php

hyper-real-time. The term *practical* implies that the method can be developed using software and hardware that is reasonably affordable by typical computer users. The method is designed to take advantage of massively parallel computations that are possible using GPU hardware. The main idea of the method is to treat ice as a set of discrete objects with a fast execution properties, and to model the system mechanics mainly as a set of discrete contact and failure events. In this way it becomes possible to parallelize the problem, so that a very large number of ice floes can be modeled. This approach is called the Ice Event Mechanics Modeling (IEMM) method which builds a system solution from a large set of discrete events occurring between a large set of discrete objects. The discrete events among the discrete objects are described with simple event equations (event solutions). It is unlike existing methods (such as finite element [64] and discrete element [40] methods, and others such as Particle in Cell [61] methods) that are built on the ideas of continuum mechanics.

With the relatively recent development of GPUs it has become possible to employ massively parallel computation on the level of a desktop computer. Massively parallel computation coupled with discrete event solutions for ice-ice and ice-structure interactions are combined to create a method to permit the rapid practical simulation of realistic ice behaviour. The approach permits the development of useful solutions to a number of practical problems that have been plaguing the designers of arctic offshore constructions (ships and structures) for many years. The problem components are as follows:

1. Discreteness and Fidelity: Almost any photograph of a ship or a structure in ice,

or a photo of the ice itself will indicate the ice is not smooth. The ice is actually a very large number of discrete, nearly rigid objects, all interacting with each other and any structure that we place in the sea. Standard approaches used to model this situation fail to capture in any realistic way the discreteness of the situation. Either the models focus all their attention in single events (single collisions) or they treat multiple events by smoothing the problem into some form of continuum. This leads to a general lack of confidence in models, and an over reliance on the scarce, expensive and inadequate full scale data. There is a great need for models that can support engineering design and assessment of arctic structures, models that will have features that are obviously and demonstrably comparable to the discrete features that are apparent in real sea ice.

2. Training: To allow for improved training of vessel operators in realistic ice conditions, we must have ship ice interaction calculations performed and displayed in real time. This is a significant challenge, due to the complexity of ice and the nature of the mechanics of solids. With most vehicles (cars, planes, ships in water), the vehicle is passing through or over a relatively smooth continuum. The environment is not altered by the vehicle. In the case of ice, the vessel must break the ice, and the ice will remain broken. (Planes do not break the air, cars do not break the road). Modeling ice loads using standard approaches (finite element modeling etc) takes so long that real-time simulation is not feasible. The IEMM approach enables a high degree of realism in training situations.

3. Long range Planning and Design: Arctic resource developments will require

> many novel ships and structures. In the past it would have been normal practice to learn from novel designs through a system of trial and error (success and failure). Increasingly there is a need to lower risks and plan against failure in advance. As such there is a need to conduct the trial and error exercises through long term high fidelity simulations, to the greatest practical extent. The IEMM concept is aimed at this challenge. By enabling hyper-real-time simulation with high physical fidelity, it will be possible to conduct design-life-length simulations, with treatment of evolving ice conditions, realistic operations and natural variability. The concept will enable designers, regulators and stakeholders in offshore projects to gain a much greater level of confidence in the safety of the projects and the key issues that must be addressed.

Graphics Processing Units (GPUs) are considered one of the most powerful form of computing hardware currently [51]. There are many researchers and developers who have become interested in using this power for general purpose computing. An overview of some applications in which general-purpose computing on graphics hardware has been successful is described in Section 2.2.

GPUs are particularly attractive for many numerical problems, not only because they provide tremendous computational power at a very low cost, but also because this power/cost ratio is increasing much faster than for traditional CPUs. A reason for this is a fundamental architectural difference: CPUs are optimized for high performance on sequential code, with many transistors dedicated to extracting instruction-level parallelism with techniques such as branch prediction and out-of-order execution.

On the other hand, the highly data-parallel nature of graphics computations enables GPUs to use additional transistors more directly for computation, achieving higher arithmetic intensity with the same transistor count [51]. Many other computations found in modeling and simulation problems are also highly data-parallel and therefore can take advantage of this specialized processing power.

Hence, this research uses the benefit of the high performance of the GPU to implement fast algorithms that can simulate ice-ice and ice-structure interactions in a very short time. Simulation of the behaviour of a ship operating in pack ice is a computationally intensive process to which a GPGPU approach can be applied. This thesis presents an efficient parallel implementation of such a simulator developed using CUDA. It also presents the results of the experiments that have used to evaluate the performance of the algorithms that have developed in this work. Moreover, it describes the experiments to validate the numerical model of ship operations in 2D pack ice and the useful applications that have been done using this simulator in planning ice management activities.

The experiments that have been used to evaluate the performance of the algorithms will be discussed in detail in Chapter 5 and are listed below:

- The first experiment explores the effectiveness of CUDA using a GPGPU approach to detect and locate the intersection between polygons. It consists of implementing both GPU and CPU versions of the algorithms to detect and locate the intersection between polygons and then running both of them on various data sets of various sizes and measuring the speed-up.

- The second experiment explores the effectiveness of CUDA using a GPGPU approach for 2D triangulation of polygons. It consists of implementing a GPU and CPU versions of the algorithm and then running both of them on various data sets of polygons of various set sizes and measuring the speed-up.

- The third experiment consists of implementing both GPU and CPU versions of the simulator and running them both on various ice fields for several iterations to compare the performance.

- The fourth experiment explores two alternative collision detection approaches. The first approach is called uniform grid [20] and the second approach is called triangulation mesh [41]. It also discusses the performance evaluation of the two approaches.

A number of experiments were done to validate the numerical model of ship operations in 2D pack ice. These experiments will be discussed in detail in Chapter 6. It consists of the design and execution of experiments to validate a Graphics Processing Unit based numerical modeling of ship operations in 2D pack ice. Using a polypropylene vessel and floes, ship-floe and floe-floe interactions are modelled in a model basin and recorded on camera. The video is processed using image processing techniques to track individual floes (and the vessel) to calculate their position and velocity over time. These results are compared with those of a numerical simulation using identical initial conditions. Conclusions are drawn about the accuracy of the numerical model and several points of improvement are identified.

The useful applications that have been done using this simulator in planning ice management activities will be discussed in detail in Chapter 6 and are listed below:

- The first application explores the use of an event-mechanics approach that is used in the GPGPU model to asses vessel performance in pack ice.
- The second application explores the use of an event-mechanics approach that is used in the GPGPU model to assess local ice loads on a vessel operating in pack ice.

## 1.1 Ice Floe Simulation

The particular problem that we are investigating is to simulate the behaviour of floating ice floes (pack ice, see Figure 1.1) as they move under the influence of currents and wind and interact with land, ships and other structures, possibly breaking up in the process. In a two-dimensional model, we model the floes as convex polygons and perform a discrete time simulation of the behaviour of these objects. The goal of this work is to be able to simulate behaviour of ice fields sufficiently quickly to allow the results to be used for planning ice management activities, and so it is necessary to achieve many times faster than real-time simulation.

## 1.2 Scope

This project is structured in two components, the *Ice Simulation Engine*, which is the focus of this thesis, and an *Ice Simulation Viewer*, which is being developed to display

Figure 1.1: Ice floes.[28]

the data produced by the simulation engine. The simulation viewer displays frames of ice field data sequentially to provide its user with a video of a simulation of the field. It is currently used by the STePS$^2$ software team to help determine the validity of the data calculated by the simulation and will eventually be used to present results to project partners. The Ice Simulation Viewer is being developed in C++ using the Qt [4] user interface framework. Figure 1.2 shows a screenshot of the main interface of the Ice Simulation Viewer with an ice field loaded. For more details about the Ice Simulation Viewer see [2].

This thesis handles the 2D simulation of pack ice and consider driving forces (e.g.,

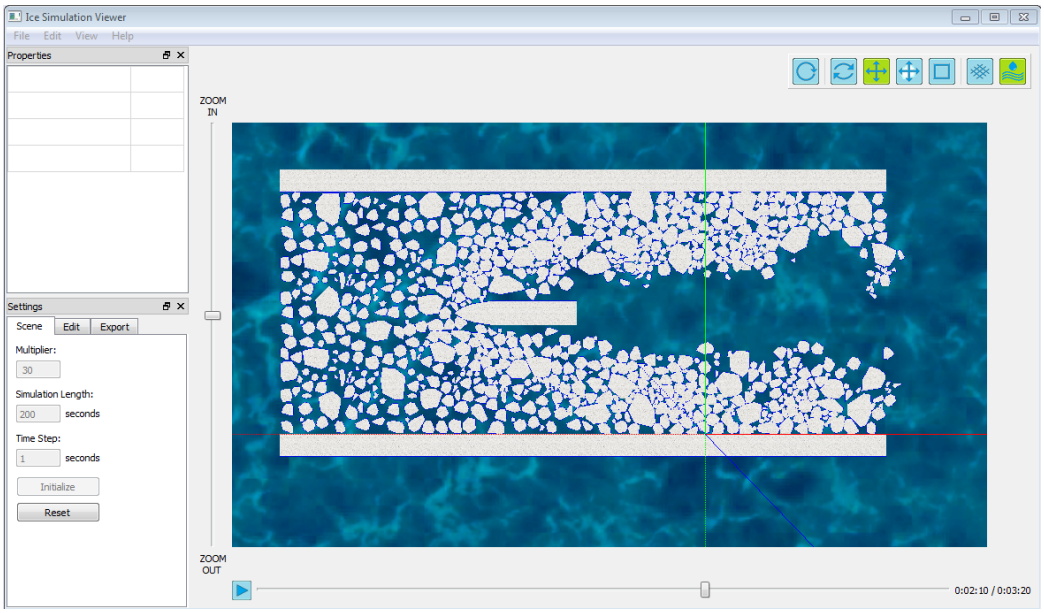



Figure 1.2: Ice simulation viewer.

current, wind) and investigates Modeling of 3D aspects but doesn't consider motion in 3D. The goal is to achieve a simulation that is fast enough to be practically used for planning ice behaviour and vessel activities in realistic size ice fields. The 3D version of the simulation will be left for future work.

## 1.3 Collaborations

In this thesis, there are a number of aspects that were done in-collaborations with Dr. Claude Daley and Roelf C. Dragt. These collaborations are discussed in detail in Chapter 6. A full list of the contributions of this thesis are discussed in detail in Chapter 7. The contributions in these collaborations are as follows:

- Design and coding of the GPGPU ice simulator software to implement the ice

mechanics equations.

- Implementing and optimizing collision detection and intersection algorithms that are used in the ice simulator to get a hyper-real-time simulation.

- Performed all experiments using various ice fields to obtain all the numerical data from the simulations that is needed to validate the GPGPU model functionality and to assess the local ice loads on a vessel operating in pack ice.

## 1.4 Outline of this Thesis

Chapter 2 describes the related work. Chapter 3 describes the mechanics of the GPGPU model. Chapter 4 describes the programming language that is used to implement the simulator, the algorithms for the collision detection and the simulator framework. The algorithm development and experiments to evaluate the performance are discussed in Chapter 5, and Chapter 6 discusses the experiments to validate the GPGPU model and it also discusses the useful application of the GPGPU ice simulator in simulating and analysis vessel operations in pack ice. Chapter 7 discusses the conclusions and future work.

# Chapter 2

# Related Work

## 2.1 Ice Simulation

The interaction between a ship and ice is a complex process. The most important factors that this process depends on are: the ice conditions, the hull geometry and the relative velocity between the ship and the ice. The main idea of ice breaking was explained by Enkvist et al [19]. Kotras et al. [32] and Valanto [62] described an overview of ship-level ice interaction where they divided the interaction process into several phases: breaking, rotating, sliding and clearing. This work focuses on the 2D clearing in open pack ice and the ice breaking.

A good understanding of the processes of ship-ice interaction is essential for developing reliable theoretical models. These models help optimise the design and operation of ships in Arctic waters. Several performance models exist, including semi-analytical and purely empirical variants, e.g. [38, 29, 55]. These models can be

used in the early design stage for an icebreaker to choose a hull form and a propulsion system that give the best possible performance in terms of global resistance, available thrust, maximum speed and fuel consumption. As well as they can be used to help ship crew optimise their route.

Lubbad et al. [39] described a numerical model to simulate the process of ship-ice interaction in real-time, using PhysX [44], a real-time physics engine middleware SDK, to solve the equations of rigid body motions for all ice floes in the calculation domain. They have validated their results of the simulator against experimental data from model-scale and full-scale tests. The validation tests showed a adequate agreement between the model calculations and experimental measurements. The goal of this work is to be able to simulate behaviour of ice fields sufficiently quickly by using GPGPU to allow the results to be used for planning ice management activities, and so it is necessary to achieve many times faster than real-time simulation. The results of that work suggest that the level of hyper-real-time performance that we hope to achieve will not result from PhysX. The physics engine also does not implement the realistic ice mechanics.

## 2.2 GPGPU Applications

GPUs have a large number of high-performance cores that are able to achieve high computation and data throughput. Currently, GPUs have support for accessible programming interfaces and industry-standard languages such as C. Hence, these chips have the ability to perform more than just the specific graphics computations

for which they were originally designed. Developers who use GPUs to implement their applications often achieve speedups of orders of magnitude vs. optimized CPU implementations [26].

There are several advantages of GPGPU that make it particularly attractive: Recent graphics architectures provide tremendous memory bandwidth and computational horsepower. Graphics hardware is fast and getting faster quickly. Graphics hardware performance is increasing more rapidly than that of CPUs because of semiconductor density, driven by advances in fabrication technology, increases at the same rate for both platforms.

Section 2.2, describes an overview of some applications in which general-purpose computing on graphics hardware has been successful.

### 2.2.1 Physically Based Simulation

There are several researchers who have developed particle system simulation on GPUs. Kipfer et al. [30] described an approach for simulating particle systems on the GPU including inter-particle collisions by using the GPU to quickly re-order the particles to determine potential colliding pairs. Kolb et al. [31] described a GPU particle system simulator that provides a support for accurate collisions of particles with scene geometry by using GPU depth comparisons to detect penetration. A simple GPU particle system example is provided in the NVIDIA SDK [22]. They described how to implement a particle system in CUDA, including particle collisions using a uniform grid data structure which will be described in chapter 5. The uniform grid

data structure to handle collisions has been tried in this work but it didn't result in better performance than the current approach that has been used in this thesis.

Several groups have used the GPU to successfully simulate fluid dynamics. A couple of papers described solutions of the Navier-Stokes equations (NSE) for incompressible fluid flow using the GPU [5, 21, 27, 33]. Harris [25] gives an introduction to the NSE and a complete description of a basic GPU implementation. Harris et al. [27] used GPU-based NSE solutions with partial differential equations (PDEs) for thermodynamics and water condensation and light scattering simulation to develop visual simulation of cloud dynamics. A simulation of the dynamics of ideal gases in two and three dimensions using the Euler equations on the GPU was described in [24].

Recent work shows that the rigid body simulation for computer games performs very well on GPUs. Havok [6, 23] explained an API for rigid body and particle simulation on GPUs which has all features for full collisions between rigid bodies and particles, and provides support for simulating and rendering on separate GPUs in a multi-GPU system. Running on a PC with dual NVIDIA GeForce 7900 GTX GPUs and a dual-core AMD Athlon 64 X2 CPU, Havok FX achieves more than a 10x speedup running on GPUs compared to an equivalent, highly optimized multithreaded CPU implementation running on the dual-core CPU alone.

The ice simulation is a computationally intensive process and consists of many interacting ice floes behaving according to physical models which make it similar to these kinds of simulation in this category.

### 2.2.2 Computational Geometry

GPUs have been found useful in computational geometry such as collision detection and Constructive Solid Geometry (CSG).

Stewart et al [59] have designed an algorithm for Overlap Graph subtraction Sequences using the GPU and explain how it can be used with the Sequenced Convex Subtraction (SCS) algorithm for CSG Rendering. The SCS algorithm for CSG sequentially subtracts convex volumes from the z-buffer (When an object is rendered, the depth of a generated pixel (z coordinate) is stored in a buffer called z-buffer). The performance of the algorithm is determined by the length of the subtraction sequence used. They have used an approach which results in faster subtraction of large numbers of convex objects from the z-buffer. Object-space intersection detection (spatial overlap) is used as a means of producing shorter subtraction sequences. They have used a term *overlap graph* to store the spatial relationship of the objects in a CSG product. Any CSG tree can be represented as a union of products termed *sum-of-products*. CSG products consist only of intersections and subtractions. Nodes in the graph correspond to shapes or objects while edges in the graph indicate spatial overlaps. Bounding volumes are used to build the overlap graph. Experimental results indicated speed-up factors of up to three.

Pascucci [52] has introduced an approach to compute isosurfaces using GPUs. Using the vertex programming capability of modern graphics cards the cost of computing an isosurface is transferred from the CPU to the GPU. Vertex programming is an assembly language interface to the transform and lighting unit in the GPU and

it provides instruction set to perform all vertex math. This has the advantage that the task is off-loaded from the CPU and storing the surface in main memory can be avoided.

# Chapter 3

# GPGPU Model Description

## 3.1 Model Input

The model can simulate all transit scenarios in which both the ice floes and the vessel are modelled as convex polygons with less than twenty sides. For numerical reasons, the bodies are not allowed to be in initial contact with each other and all objects and processes are two dimensional.

The floes in the model have three degrees of freedom: movement in x-and y-direction and rotation around the z-axis. The ship is restricted to one degree of freedom, movement in x-direction (forward movement). Figure 3.1 shows the 2D concept.

The starting position of all the floes and the vessel are stored in an .ice-file. This file type is used as the input for the numerical simulation and contains all the positions and initial velocities of the bodies (vessel, floes and sides). For more details about

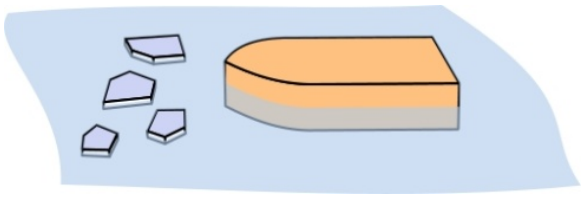


Figure 3.1: Sketch of 2D concept used in simulations. [11]

the file structure see Appendix B.

## 3.2 Model Mechanics

### 3.2.1 Ice Behaviour

As stated in Chapter 1, the concept for the simulation is the rapid assessment of a sequence of discrete interactions with a large number of discrete ice objects. The transit of a vessel through pack ice, and the interactions of the ice are modeled as a set of contact events. The movements are treated using simple equations of motion. The individual ice blocks move in the 2D space of the simulation. The position and velocity of each floe is updated every time step. A simple water drag model results in the floes tending to slow. Ice-ice interactions account for both ice crushing impact forces and steady elastic stresses to resist static pressure. In this generation of the model there are no rafting, rubbling and no floe splitting. These are being planned for future generations of the model.

Each ice-ice collision event within the pack is treated using a method that can be traced to Popov et. al [53]. The method was updated to reflect pressure-area effects [9], and used for a variety of ship-ice interaction scenarios [12]. When two

bodies collide in a 2D world, each body has 3 degrees of freedom, as well as two mass parameters, and a shape (see Figure 3.2). The large number of parameters makes the collision problem potentially very difficult. The problem can be substantially simplified by making a few simplifying assumptions and viewing the problem from the perspective of the collision point. It is assumed that the collision will be of short duration, and that the force will act, in the frictionless case, normal to the line of contact (see Figure 3.3). With these assumptions the problem can be reduced to an equivalent one dimensional collision. The equivalent velocity is the closing velocity at the point of contact along the collision normal

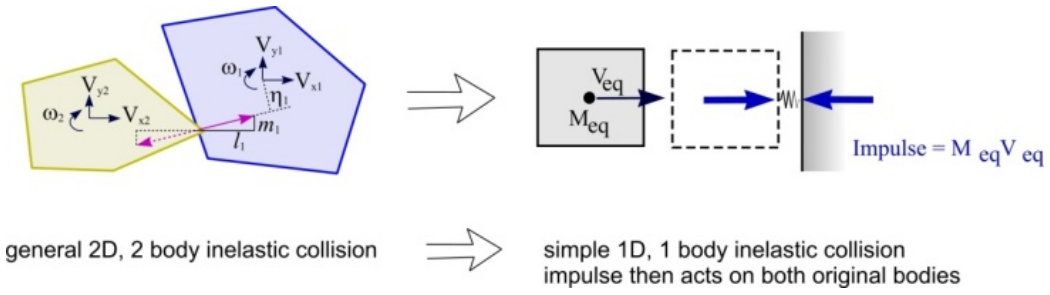



Figure 3.2: Idealization of 2D collision between two finite bodies. [10]

The mass reduction factor ($R$) for one body subject to a collision along a normal is:

$$R = l^2 + m^2 + \frac{\eta^2}{r_x^2} \tag{3.1}$$

Where $l$ and $m$ are direction cosines of the inward normal vector, $\eta$ is the moment arm of the normal vector about the centroid and $r_x^2$ is the square of the radius of gyration of the body (see Figure 3.2). Each body in a two body collision has a unique mass reduction factor. The above mass reduction factor represents the simplest case

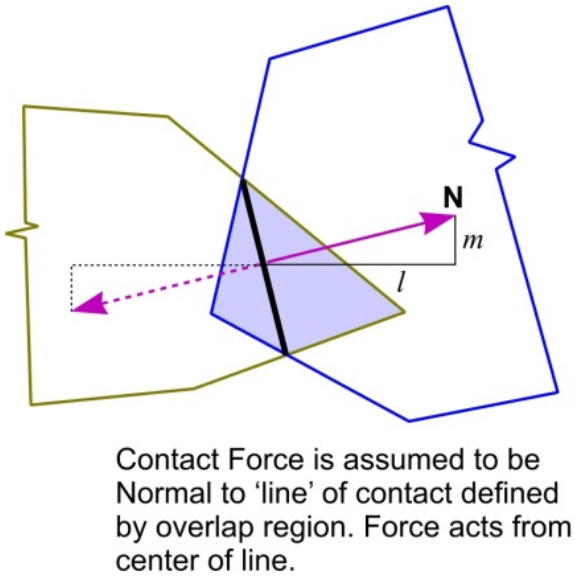



Figure 3.3: Assumption concerning the location and direction of impact forces. [10]

for 2D without added mass or friction. Enhancements to the formula have been developed to include effects of hydrodynamic added mass and friction and 3D effects (see [9]).

The program assumes that all collisions are inelastic, where the ice crushing energy absorbs all the effective kinetic energy. A collision is detected in one time step when the two bodies are found to overlap. The effective masses and normal velocities are determined for each colliding body for their respective points of impact. The direction of relative motion is determined to allow the determination of the friction direction. The impulse that will eliminate the net normal velocity is then found. That impulse is applied to each body in an equal and opposite sense. The result is that the normal velocity at that point is zero in the next time step. This does not mean that all motion is stopped. Ice floes tend to rotate around the collision point and slide away.

This approach does contain some idealizations and approximations, but does appear to be stable and produce reasonable results.

The forces are found by using the "process pressure-area" relationship for ice, the ice edge shape, hull angles, and effective mass of each collision (see [9]). It should be noted that two distinct versions of this approach are used in the Ice-Event-Mechanics simulation. The kinematics of the vessel and ice are modeled in 2D, so one implementation of the model derives the 2D forces. Those algorithms assume that the vessel is wall sided, and do not permit ice to move under the hull. Another algorithm takes the hull form into account and determines impact forces using the 3D mechanics and shapes. These 3D forces are logged for later analysis. For the above reasons, the simulation presented is termed a 2.5D simulation. It is for this reason that the simulations are limited to open pack. High ice concentrations and pressure in the ice pack would create conditions that would invalidate the assumptions. Future model development is planned to remove these restrictions. For more details about the GPU ice simulation equations see Appendix A.

### 3.2.2 Vessel Description

This section describes the vessel that was used in the experiments in Chapter 6. The vessel currently simulated is 100m long and 20m wide. The vessel is meant to represent a large offshore supply vessel with some ice capability. In plan view, the vessel's waterline is a polygon as shown in Figure 3.4. The bow of the vessel is sloped as an ice-going vessel would be. Figure 3.5 shows the 3D shape of the vessel.

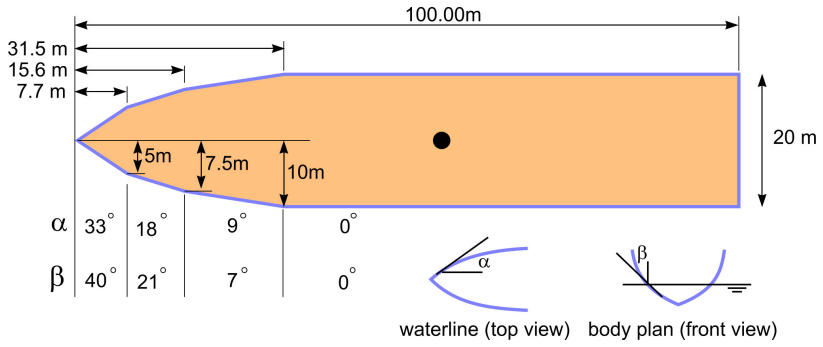



Figure 3.4: Geometry of 2D vessel polygon. [10]

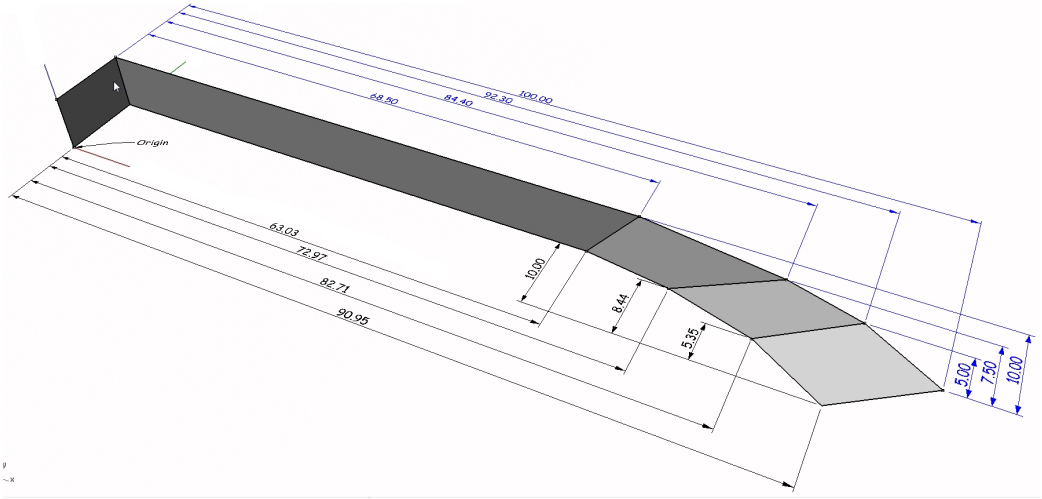



Figure 3.5: 3D shape of the vessel (half full shown). [10]

The vessel moves through the ice pack using a simple auto pilot model, rather than at a fixed speed and direction. There is a constant-power thrust and water resistance model which combines the effects of a reduction in net vessel resistance and an increase in propeller thrust as the vessel is slowed. In the absence of the pack ice, this net thrust model brings the vessel to a steady forward speed from a standing or moving start.

The net thrust is calculated as follows:

$$T_{net} = T_{bollard} - C_{resistance}V^3 \tag{3.2}$$

Where $T_{net}$ is the net thrust applied to the vessel model, $T_{bollard}$ is the arbitrary assigned bollard (zero speed) thrust and $V$ is the ship velocity.

The constant $C_{resistance}$ incorporates both resistance reduction and thrust increase effects and is calculated for each bollard thrust such that the net thrust is zero at a given open water speed $V_{ow}$. This model means that the vessel has a declining net force applied to it as the speed increases and will find a lower equilibrium speed as the average ice force from ice impacts increases. The simulations that were done in Chapter 6 cover 5 power levels, which are expressed in terms of the bollard thrust from a low of 46.25kN to a high of 740kN of thrust.

When the vessel strikes an ice floe it can be slowed or deflected or both. Course control is achieved by providing un-coupled heading and sway Proportional-Derivative controls that apply a countering sway force and a countering yaw moment when deviations in the set heading and course line are detected. Damping is provided by sway and yaw velocity dependent terms.

$$M_{yaw} = G_1\delta\theta + G_2\frac{d\theta}{dt} \tag{3.3}$$

$$F_{sway} = G_3\delta y + G_4\frac{dy}{dt} \tag{3.4}$$

Where $M_{yaw}$ is the correcting moment, $\delta\theta$ is the deviation from the set heading, $F_{sway}$ is the correcting sway force, $\delta y$ is the deviation from the set track and

$G1, G2, G3, G4$ are controller gains that are set to achieve the desired course holding characteristics.

This simple autopilot steers the vessel back on track. In this way the vessel more realistically responds to the multiple collisions that it experiences. Floe impacts tend to slow the vessel and cause deviations in the track and heading but these deviations are countered by the change in thrust or changes in moment and sway force. For more details about the properties of the vessel see Appendix A.1.2.

# Chapter 4

# Methodology

## 4.1 Single Program Multiple Data (SPMD) Parallel Programming Model

This work uses the SPMD programming model. In this programming model all tasks perform the same computations on different partitions of the data. This model also has the required features to allow the tasks to branch or conditionally perform some of the computations. Therefore, it is not necessarily that all tasks should perform all computations. Figure 4.1 illustrates the SPMD model.

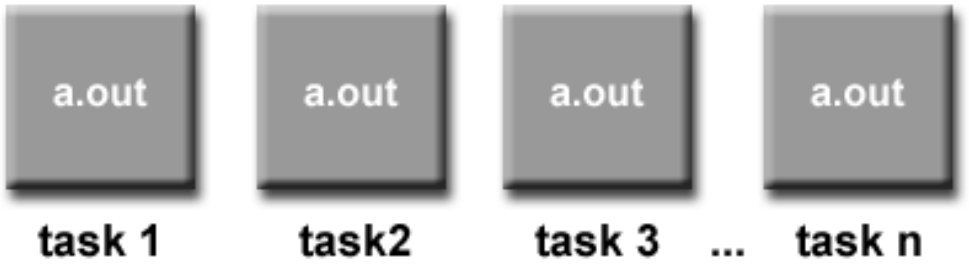



Figure 4.1: SPMD model. [3]

## 4.2 Stream Processing

The basic programming model of traditional GPGPU is stream processing, which is closely related to SIMD[1]. A uniform set of data that can be processed in parallel is called a stream. The stream is processed by a series of instructions, called a kernel [50]. Stream processing is a very simple and restricted form of parallel processing that avoids the need for explicit synchronization and communication management. It is especially designed for algorithms that require significant numerical processing over large sets of similar data (data parallelism) and where computations for one part of the data only depend on 'nearby' data elements. In the case of data dependencies, recursion or random memory accesses, stream processing becomes ineffective [50, 7]. Computer graphics processing is very well suited for stream processing, since vertices, fragments and pixels can be processed independently of each other, with clearly defined directions and address spaces for memory accesses. The stream processing programming model allows for more throughput-oriented processor architectures. For

[1]Single Instruction Multiple Data, in the Flynn's taxonomy of computer architectures

example, without data dependencies caches, can be reduced in size and the transistors can be used for ALUs instead. Figure 4.2 shows a simple model of a modern CPU and a GPU. The CPU uses a high proportion of its transistors for controls and caches while the GPU uses them for computation (ALUs).

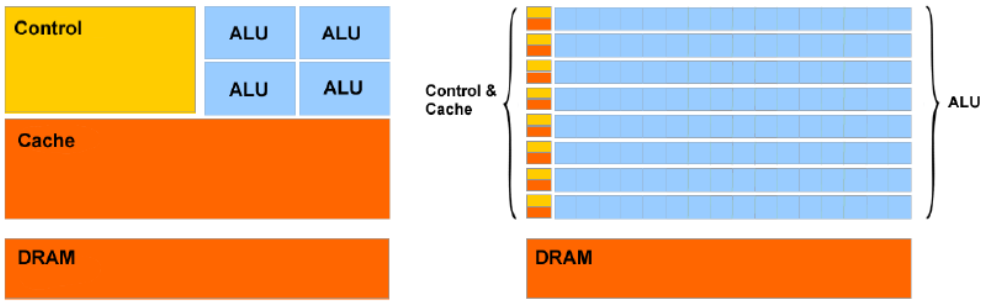



Figure 4.2: Simple comparison of a CPU and a GPU. [47]

## 4.3 CUDA

Compute Unified Device Architecture (CUDA) is a comprehensive software and hardware architecture for GPGPU that was developed and released by Nvidia in 2007. It is Nvidia's move into GPGPU and High-Performance Computing (HPC), combining good programmability, performance, and ease of use. A major design goal of CUDA is to support heterogeneous computations in a sense that serial parts of an application are executed on the CPU and parallel parts on the GPU[46]. A general overview of CUDA is illustrated in Figure 4.3.

Nowadays, there are two distinct types of programming interfaces supported by CUDA. The first type is using the device level APIs (left part of Figure 4.3) in

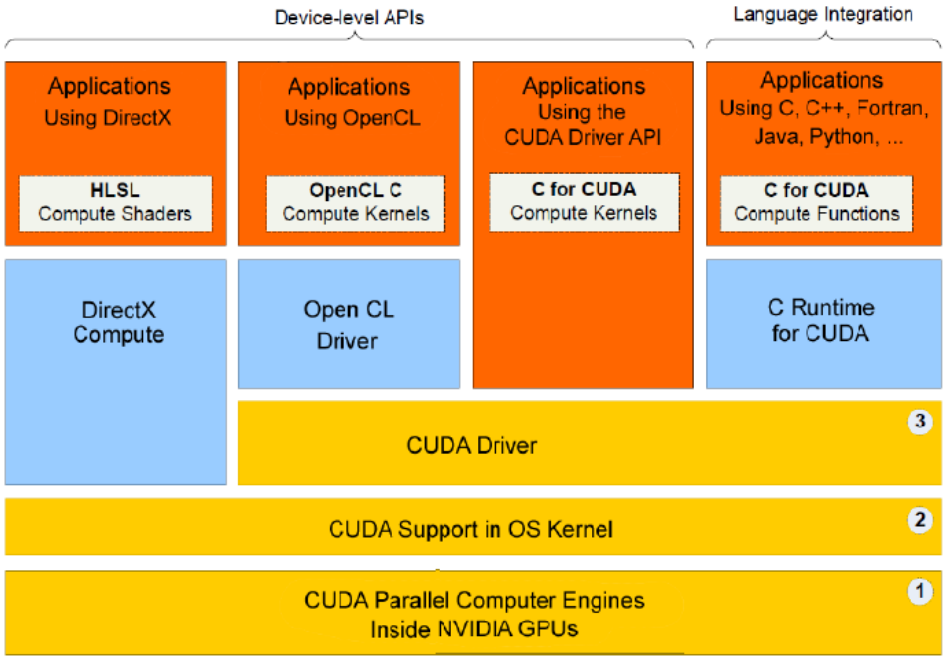



Figure 4.3: CUDA overview. [45]

which we could use the new GPGPU standard DirectX Compute by using the high-level shader language (HLSL) to implement compute shaders. The second standard is OpenCL which is created by the Khronos Group (as is OpenGL). OpenCL kernels are written in OpenCL C. The two approaches don't depend on the GPU hardware; hence they can use GPUs from different vendors. In addition to that, there is a third device-level approach through low-level CUDA programming which directly uses the driver. One advantage of this approach is it gives us a lot of control, but a disadvantage is that it is complicated because it is low-level (it interacts with binaries or assembly code). Another programming interface is the language integration programming interface (right column of Figure 4.3). It is better to use the C runtime for CUDA, which

is a high-level approach that requires less code and is easier in programming and debugging [45]. Also, other high-level languages could be used (e.g. Fortran, Java, Python, or .NET languages through bindings). Therefore, this work uses the C runtime for CUDA.

Based on [43], the CUDA programming model suggests a helpful way to solve the problems by splitting it into two steps: First dividing the problem into coarse independent sub-problems (grids) and then into finer sub-tasks that can be executed cooperatively (thread blocks). The programmer writes a serial C-for-CUDA program, which invokes parallel *kernels* (functions written in C). The kernel is usually executed as a grid of *thread blocks*. In each block the threads work together through barrier synchronization and they have access to a shared memory that is only visible to the block. Each thread in a block has a different *thread ID*. Each *grid* consists of independent blocks. Each block in a grid has a different *block ID*. Grids can be executed either independently or dependently. Independent grids can be executed in parallel provided that we have a hardware that supports executing concurrent grids. Dependent grids can only be executed sequentially. There is an implicit barrier that ensures that all blocks of a previous grid have finished before any block of the new grid is started.

## 4.4 Collision Detection

Since a discrete time simulation has been used in this work, for each time step, collisions are detected by searching for regions of overlap between ice floes, computing

the force that would result from such a collision and adjusting the velocity of each floe accordingly. The problem of detecting collisions between ice floes is broken down into two parts: determining if the floes are overlapping and computing the region of overlap.

To determine whether or not two convex polygons are intersecting, the method of separating axes [17] has been used. This method is for determining whether or not two stationary convex objects are intersecting. This method is a fast generic algorithm that can remove the need to have collision detection code for each type pair (any type of convex polygons: 3-sided, 4-sided, 5-sided, etc...) thereby reducing code and maintenance.

In this method the test for nonintersection of two convex objects is simply stated: If there exists a line for which the intervals of projection (the lowest and highest values of the polygon projection on the line) of the two objects onto that line do not intersect, then the objects do not intersect. Such a line is called a separating line or, more commonly, a separating axis.

For a pair of convex polygons in 2D, only a finite set of direction vectors needs to be considered for separation tests: the normal vectors to the edges of the polygons. The left picture in Figure 4.4 shows two nonintersecting polygons that are separated along a direction determined by the normal to an edge of one polygon. The right picture shows two polygons that intersect (there are no separating directions).

Once it is determined that two polygons are overlapping, the region of overlap is identified to compute the resultant momentum. Finding the intersection of two

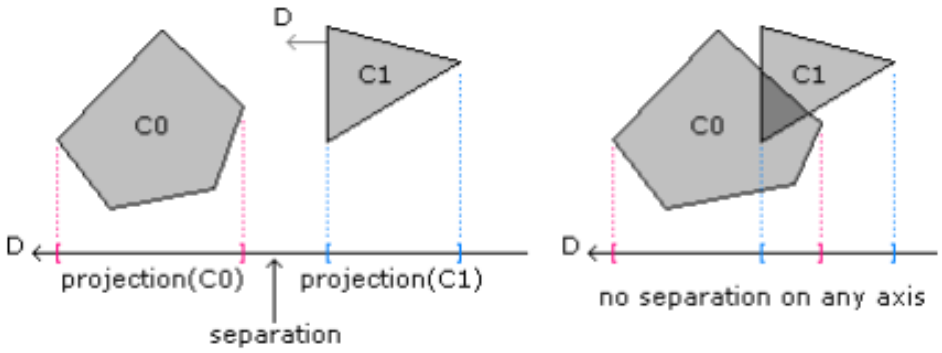



Figure 4.4: Nonintersecting convex polygons (left). Intersecting convex polygons (right). [17]

arbitrary polygons of $n$ and $m$ vertices must have quadratic complexity, $\Omega(nm)$. But the intersection of two convex polygons has only linear complexity, $O(n+m)$. Intersection of convex polygons is a key component of a number of algorithms, including determining whether two sets of points are separable by a line. The first linear algorithm was found by Shamos [57], and since then a variety of different algorithms have been developed, all achieving $O(n+m)$ time complexity [49]. This work uses the algorithm that was developed by O'Rourke, Chien, Olson & Naddor.

The basic idea of the algorithm is as illustrated in Algorithm 1 [49]. Here, the boundaries of the two polygons $P$ and $Q$ are oriented counterclockwise, and let $A$ and $B$ be directed edges on each. The algorithm has $A$ and $B$ chasing one another. An example that explains the algorithm is illustrated in Figure 4.5. The edges $A$ and $B$ are shown as vectors. For more details about the algorithm see [49].

**Algorithm 1** :Intersection of convex polygons

// *A*ssume that $P$ and $Q$ overlap

Choose $A$ and $B$ arbitrarily.

**repeat**

**if** $A$ intersects $B$ **then**

The point of intersection is a vertex.

One endpoint of each of $A$ and $B$ is a vertex.

**end if**

Advance either $A$ or $B$, depending on which is pointing at the other.

**until** both $A$ and $B$ cycle their polygons

**if** no intersections were found **then**

One polygon must be entirely within the other.

**end if**

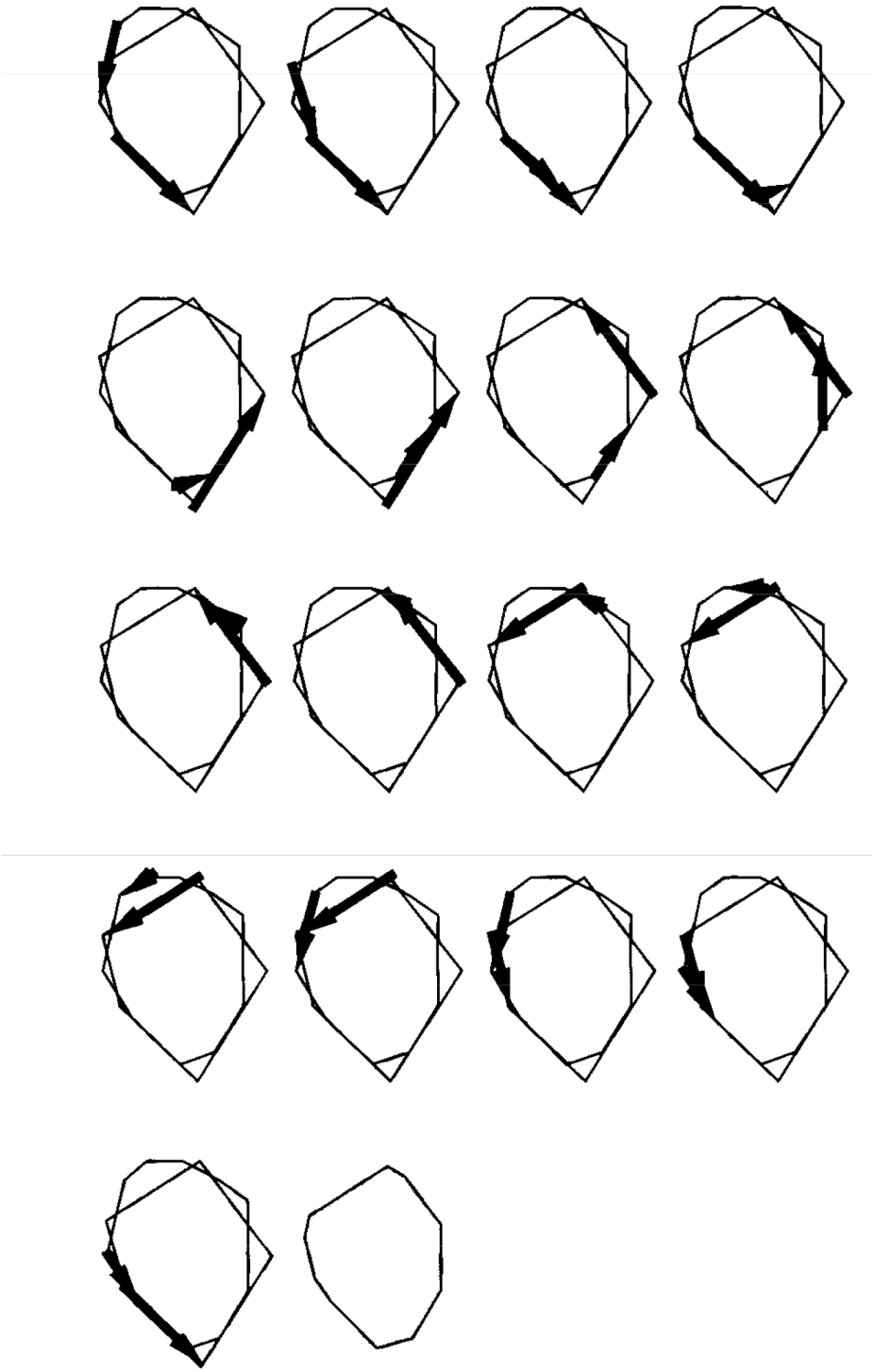


Figure 4.5: Snapshots of polygon intersection algorithm, sequenced left to right, top to bottom. [49]

## 4.5 High Level Algorithm of the Simulator

Figure 4.6 shows the high-level flow of the ice simulator. At the beginning the CPU reads the ice floe data (position and velocity) and initializes the simulation parameters. The initial data is transferred from the CPU to the GPU. Then, the GPU takes over the main work of the simulation. First, the "create neighbours list" kernel is launched to find the list of ice floes that might overlap with each ice floe. Then, the "test intersection and find collision response" kernel is launched to determine the list of ice floes that have overlap with each ice floe and to calculate the collision response for each ice floe. Last, the "update" kernel is launched to update the position and velocity for all ice floes. After that, the ice floes data is transferred back to the CPU. This process is repeated until the simulation is completed. The kernels were executed using one thread for each ice floe.

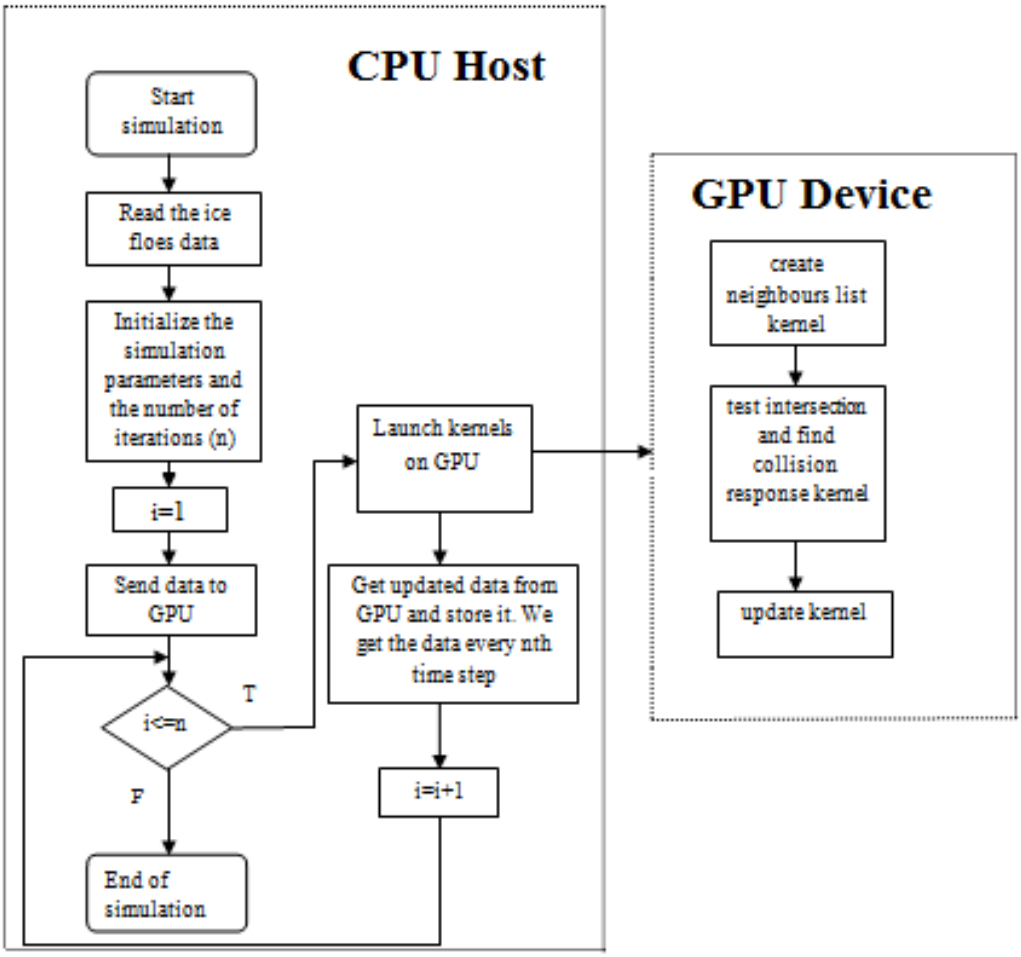



Figure 4.6: Ice simulator flowchart.

# Chapter 5

# Optimization and Development

This chapter describes the experiments that have used to evaluate the performance of the algorithms that have developed in this work. It also discusses the different approaches for implementing the ice simulator.

This work uses Intel(R) Xeon(R) CPU @2.27GHz and a GPU Tesla C2050 card which is shown in Figure 5.1. This card has 448 processor cores, 1.15 GHz processor core clock and 144 GB/sec memory bandwidth.

## 5.1 Implementation Discussion of the Ice Simulator

As implementation have been developed. Three different general structures of the GPU solution have been progressed through. They are explained below and the relative performance of these is illustrated in Figure 5.2.


Figure 5.1: Tesla C2050. [48]

In the first implementation, two CUDA kernels were used: The first kernel was executed using one thread per ice floe, it finds the list of all pair-wise collisions by determining which pairs of ice floes are overlapping. The second kernel was executed using one thread per overlapping ice floe pair, it computes the collision response (momentum) for each pair. This approach resulted in speed-up of up to 10 times as compared with the CPU implementation, but didn't achieve hyper-real-time results in all cases and therefore was insufficient.

In the second implementation the two kernels were merged in one kernel. One thread for each polygon to check the collision with other ice floes and calculate the response. This approach was a little faster than the first, but still insufficient for the general case.

In the third implementation took advantage of the fact that ice floes that are widely separated are unlikely to overlap any time soon, and therefore the number of ice floes to be checked for collision can be dramatically reduced by eliminating

those that are beyond some distance away. To do this another kernel was added to find the list of neighbours for each ice floe that are within the region where they might overlap with it soon. Therefore, instead of checking the collisions with every other ice floe, the collisions just need to be check with those in this list. The list is re-created periodically, but not every time step, so that the total number of computations is significantly reduced. This approach is significantly faster than the other two approaches as seen in Figure 5.2 and achieves substantially better than real-time for small ice fields.

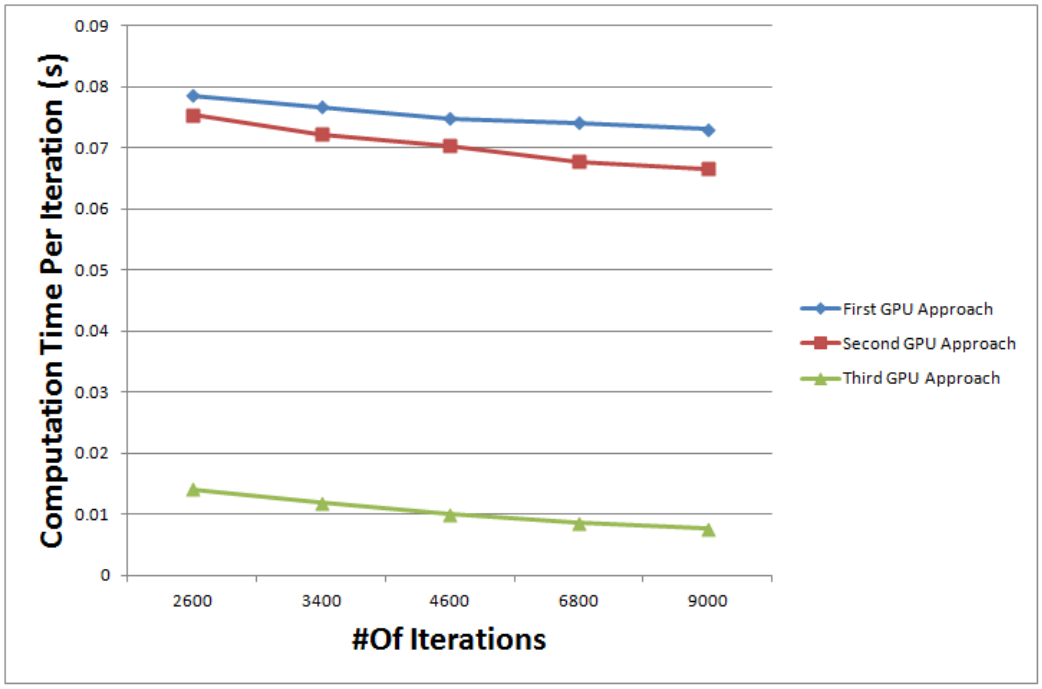



Figure 5.2: Computation time per iteration of the three GPU approaches for an ice field has 456 floes.

## 5.2 Polygon Intersection

The problem was explored here is to detect and locate the intersection between polygons. Both serial and parallel algorithms to compute the intersection between polygons were implemented. Then, both algorithms were run using 25 data sets of polygons: five different set sizes (100, 500, 1000, 2000, 3000) and each size has five data sets. These sets were generated randomly using matlab. Finally, the speed-up (ratio of time for serial algorithm to that for parallel algorithm) was measured.

### 5.2.1 Results

Figure 5.3 shows the CPU and GPU time to detect and locate the intersection between polygons for all five data sets. As seen in Figure 5.3, it is clear that the GPU time is less than the CPU time, and as the number of polygons increases, the CPU time gets much higher than GPU time. Therefore, the GPU is faster when there are a huge number of polygons.

Figure 5.4 shows the speed up in all five different cases. As seen in Figure 5.4, the highest speed up is when the number of polygons is 448. This is due the number of processor cores (448) on the GPU card that has been used in this work. Each polygon is handled by one core, but in cases where there are more than 448 polygons, one core must handle more than one polygon.

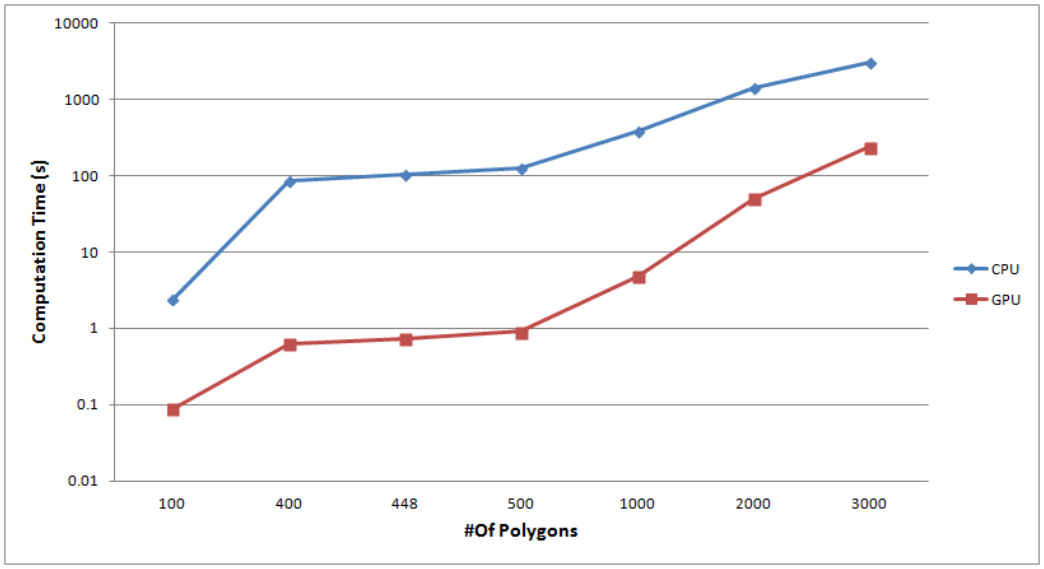



Figure 5.3: Compute time.

## 5.3 2D Triangulation of Polygons

The triangulation mesh to handle the collision detection has been tried in this work. Therefore, the present work explores the effectiveness of CUDA using GPGPU approach for 2D Triangulation of Polygons. An experiment to measure the performance of the GPU with respect to the CPU was conducted. The experiment consists of implementing a serial and parallel algorithm to triangulate 2D polygons. Both algorithms were run using 6 data sets of polygons of different set sizes (500, 1000, 2000, 4000, 8000, 16000) and then the speed-up was measured.

### 5.3.1 Polygon Triangulation by Ear Clipping

Polygon Triangulation is the process of decomposing a simple polygon into triangles. It is known from computational geometry that any triangulation of a simple polygon

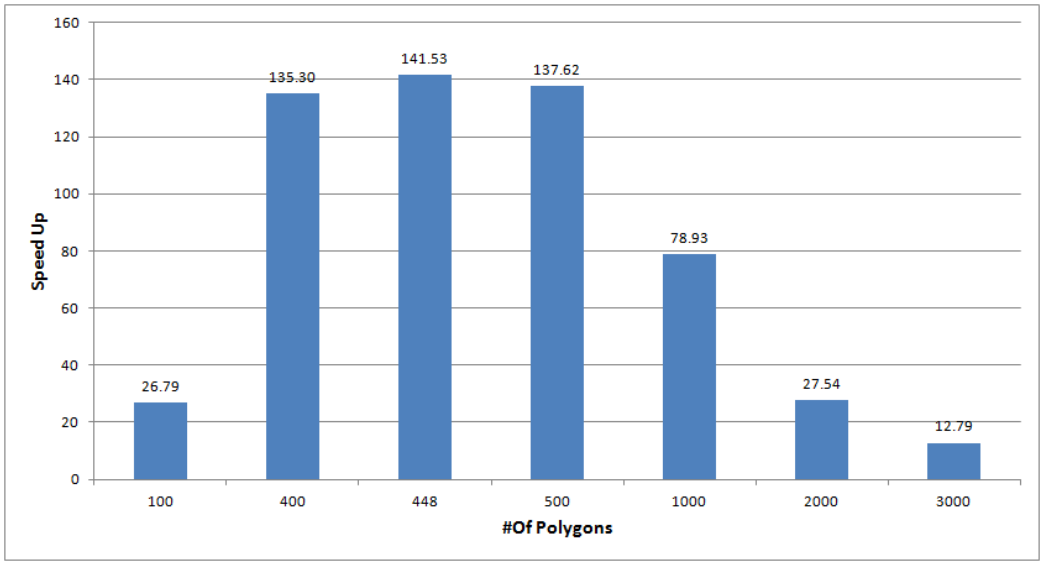



Figure 5.4: Speed up of the GPU implementation to detect and locate the intersection between polygons over the CPU implementation.

of $n$ vertices has $n-2$ triangles. There are several algorithms that have been used for polygon triangulation. In this work, the *ear clipping* algorithm have been used, which has quadratic complexity, $O(n^2)$, in the number of verticies. Another algorithm that has a linear complexity, $O(n)$, is known in theory [8] but it is more difficult to implement.

An *ear of a polygon* is a triangle formed by three consecutive vertices $V_0, V_1, V_2$ such that no other vertices of the polygon are located inside the triangle. The line segment between $V_0$ and $V_2$ is called a *diagonal* of the polygon. The vertex $V_1$ is called the *ear tip*. Based on [41], any simple polygon with at least four vertices has at least two non-overlapping ears. Therefore, the basic idea of this algorithm as illustrated in Algorithm 2 is to find such an ear, remove it from the polygon and repeat this process until there is only one triangle left.

**Algorithm 2** :Ear clipping

1. while $n > 3$ do

   (a) Locate an ear tip $v2$

   (b) Output the the triangle $v1, v2, v3$

   (c) delete $v2$

Figure 5.5 shows an example that explains the triangulation algorithm.

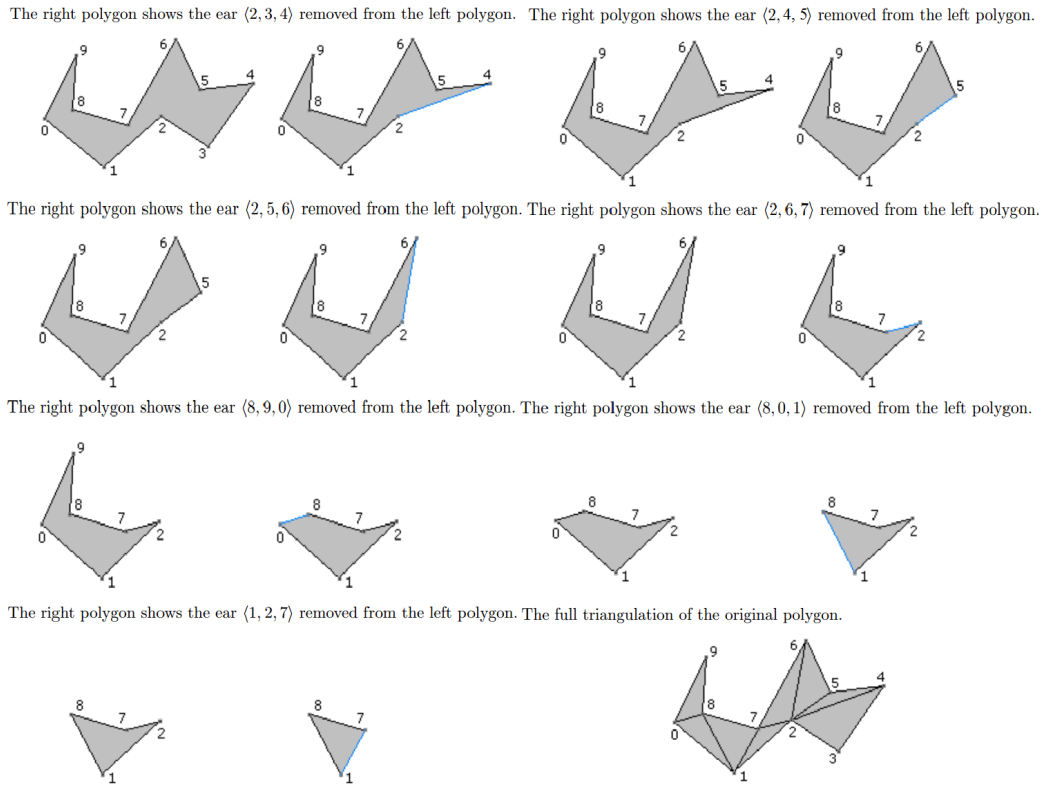



Figure 5.5: Ear clipping process. [18]

### 5.3.2 Results

Figure 5.6 shows the CPU and GPU time to triangulate polygons for all six data sets. As seen in Figure 5.6, it is clear that the GPU time is significantly less than the CPU time, and as the number of polygons increases, the CPU time gets much higher than GPU time. Therefore, the GPU is faster when there are huge number of polygons. Figure 5.7, which shows the speed up for all six set sizes. As seen in Figure 5.7, the highest speed up is when the number of polygons is 2000. This is due to the number of multiprocessors on the card that we have used (14). In CUDA each thread block executes on one multiprocessor. In this work a block size of 128 threads was used. Therefore, the number of blocks is 16 (2000/128) which is approximately equal to the number of multiprocessors (14). So, each block is approximately handled by one multiprocessor, but in cases where are more than 2000 polygons one multiprocessor must handle more than one block.

## 5.4 Ice Simulator Performance Evaluation

A serial and parallel version of the simulator were implemented and both versions were tested on two different ice fields and different (real-time) durations. The first ice field has 456 ice floes and the second one has 824 ice floes. These two ice fields will be described in Chapter 6. The simulations were run with and without ice breaking. In both cases, there was no difference in the performance therefore it didn't slow down the speed-up of the simulator. The computation time step ($\Delta t$) that was used

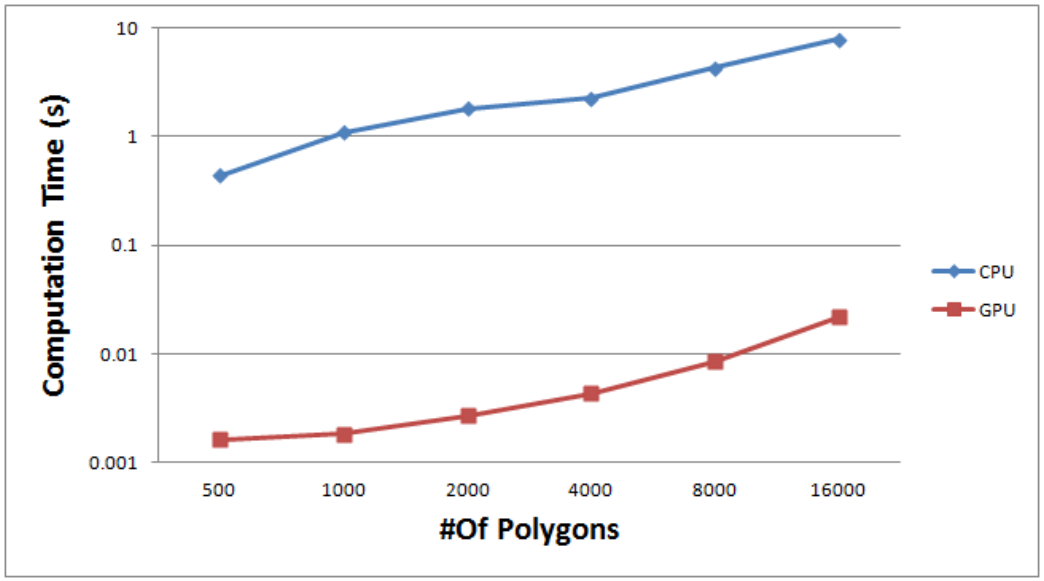



Figure 5.6: Compute time of polygon triangulation.

in the simulations is 0.1s. This time step is chosen to maintain accuracy in the ice mechanics. Two different approaches were tried to generate the list of neighbours for each ice floe: In the first approach, a fixed radius around each floe in the entire ice field was used. In the second approach, the list of neighbours were found using a variable radius specific to each floe pair (bounding circle check - see Figure 5.8). When using bounding circle method, a circle is assumed around the ice floe. The radius of this circle is equal to the largest radius in the ice floe. When an imaginary circle touches or overlaps another imaginary circle (around the ice floe), the ice floe is considered a neighbour. Finally, the speed-up was measured.

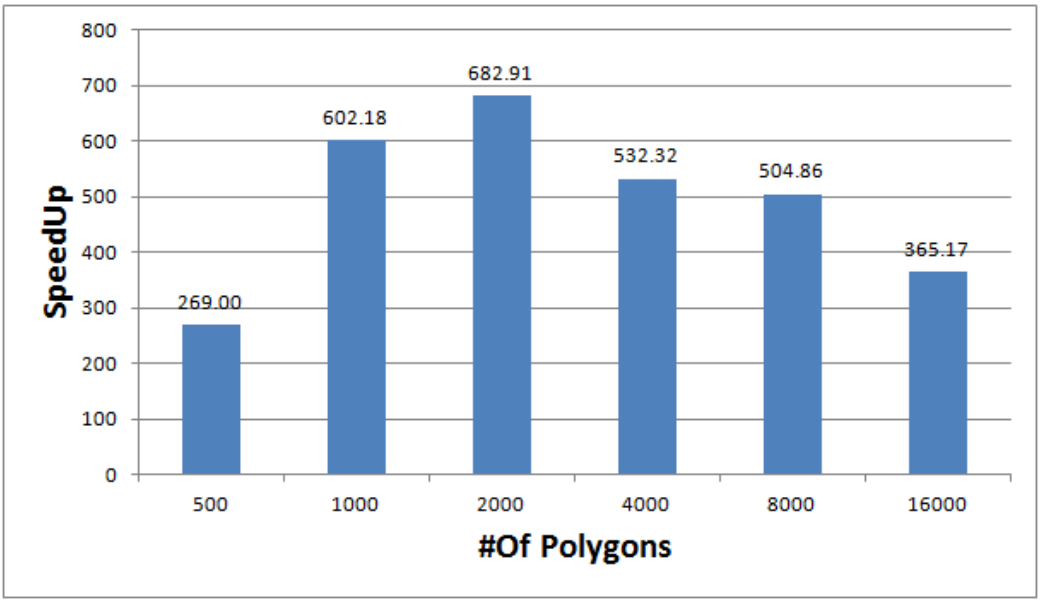



Figure 5.7: Speed up of the GPU implementation of polygon triangulation over the CPU implementation.

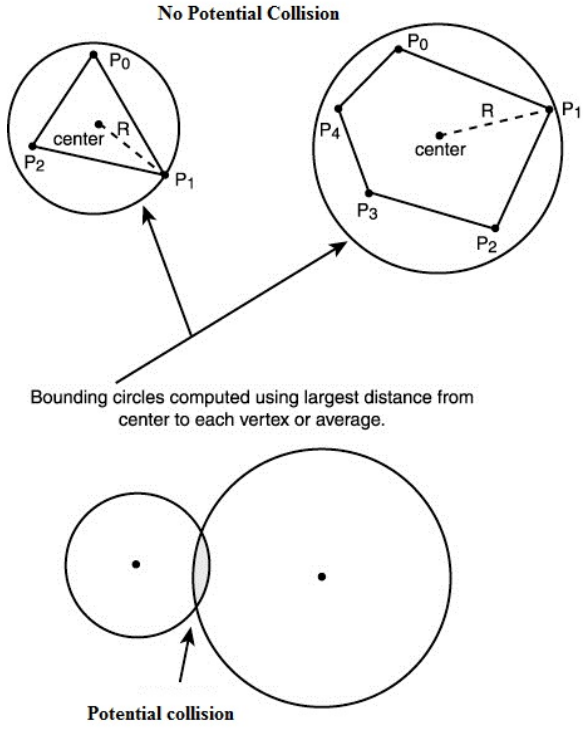



Figure 5.8: Bounding circle method.

### 5.4.1 Results

Figure 5.9 shows the CPU and GPU computation time per iteration to simulate the behaviour of the ship in the first ice field which has 456 ice floes for all five different durations (numbers of iterations) using the two approaches for generating the list of neighbours. As seen in Figure 5.9 it is clear that the second approach, using a variable radius, is faster than the first approach, using a fixed radius, and the GPU time is much lower than the CPU time. Therefore, this work uses the second approach to generate the list of neighbours. Moreover, the simulation is hyper-real-time since the computation time per iteration is less than the computation time step ($\Delta t = 0.1s$).

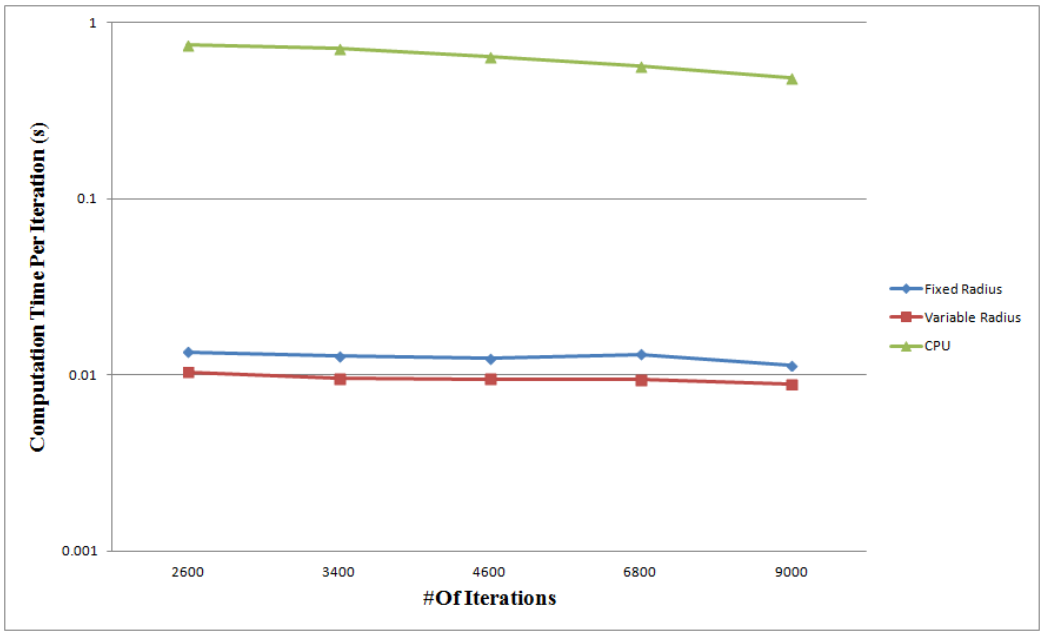



Figure 5.9: Computation time per iteration for the 456 ice field.

Figure 5.10 shows the speed up in all five different cases using the second approach for generating the list of neighbours for the first ice field.

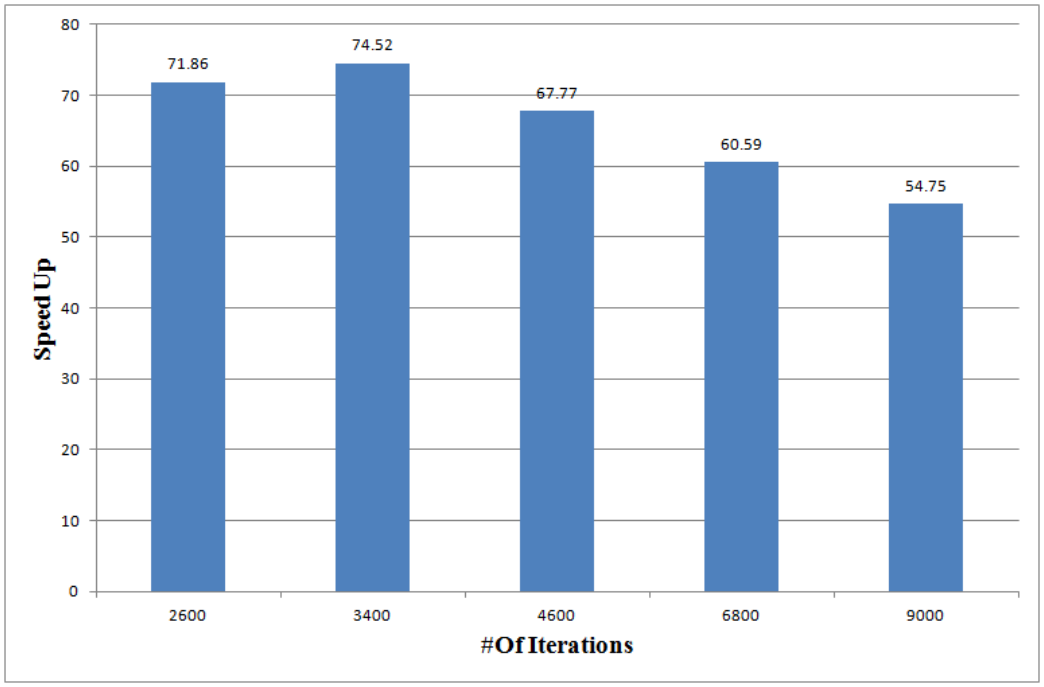



Figure 5.10: Variable radius approach speed up for the 456 ice field.

Figure 5.11 shows the CPU and GPU computation time per iteration to simulate the behaviour of the ship in the second ice field, which has 824 ice floes for all five different durations using the two approaches that we have used to generate the list of neighbours. As seen in Figure 5.11 it is clear again that the variable radius approach is faster than the fixed radius approach, and the GPU time is much lower than the CPU time. Moreover, the simulation is hyper real-time, since the computation time per iteration is less than the computation time step ($\Delta t = 0.1s$).

Figure 5.12 shows the speed up in all five different cases using the second approach for generating the list of neighbours for the second ice field.

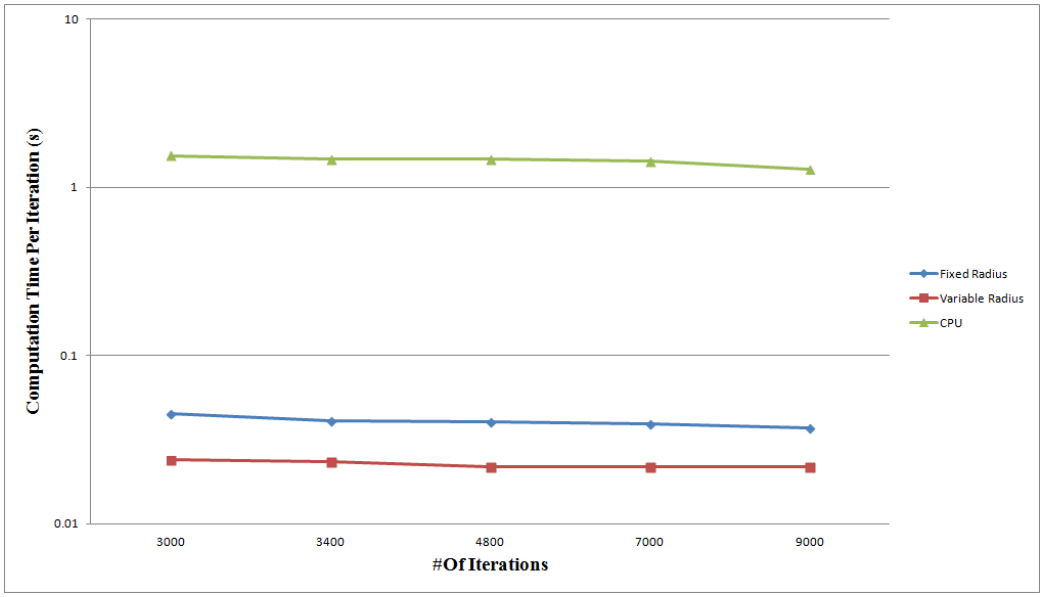



Figure 5.11: Computation time per iteration for the 824 ice field.

## 5.5 Alternative Collision Detection Approaches

This section describes an alternative collision detection approaches that have been tried in this work. It also discusses the performance evaluation of the approaches.

### 5.5.1 Uniform Grid Data Structure

In the uniform grid [20], a grid subdivides the simulation space into a grid of uniformly sized cells. For the sake of simplicity, a gird where the cell size is the same as the size of the largest ice floe (double its radius) was used. Also, the grid is called "loose" grid, where each ice floe is assigned to only one grid cell based on it is centroid. Since each ice floe can potentially overlap several grid cells, this means that the ice floes in the neighbouring cells (9 cells in total in 2D grid) must be also examined in the

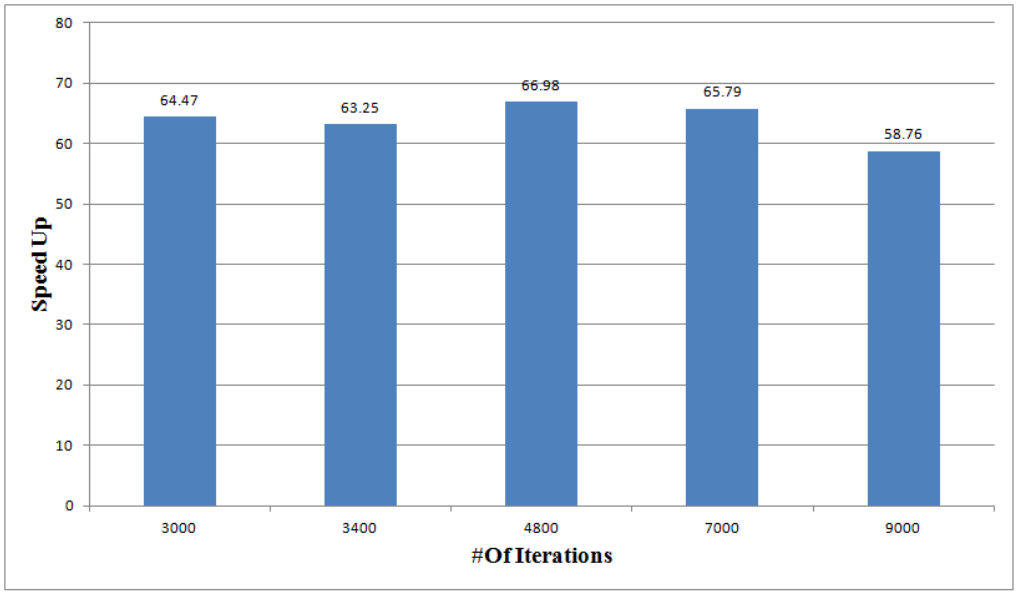
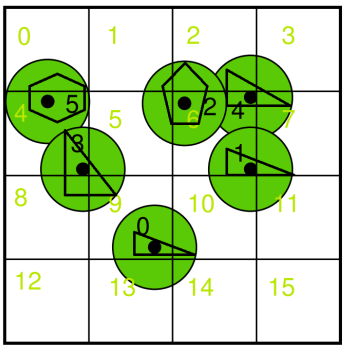



Figure 5.12: Variable radius approach speed up for the 824 ice field.

collision processing to see if they are touching the ice floe.

The grid is built using sorting. The algorithm to build the grid consists of several kernels. The first one "calcHash" calculates a hash value for each ice floe based on its cell id. The linear cell id as the hash was used. The kernel stores the results to the "particleHash" array in global memory as a uint2 pair (cell hash, ice floe id). Then, the ice floes are sorted based on their hash values. The sorting is performed using the fast radix sort provided by the CUDPP library, which uses the algorithm described in [56]. This creates a list of ice floe ids in cell order. In order for this sorted list to be useful, the start of any given cell in the sorted list must be calculated. This is done by running another kernel "findCellStart", which uses a thread per ice floe and compares the cell index of the current ice floe with the cell index of the previous ice floe in the sorted list. If the index is different, this indicates the start of a new

cell, and the start address is written to another array using a scattered write. Also, the index of the end of each cell is found in a similar way. Figure 5.13 demonstrates creating the grid using the sorting method.




| Index | Unsorted list (cell id, particle id) | List sorted by cell id | Cell start |
|---|---|---|---|
| 0 | (9, 0) | (4, 3) | |
| 1 | (6, 1) | (4, 5) | |
| 2 | (6, 2) | (6, 1) | |
| 3 | (4, 3) | (6, 2) | |
| 4 | (6, 4) | (6, 4) | 0 |
| 5 | (4, 5) | (9, 0) | |
| 6 | | | 2 |
| 7 | | | |
| 8 | | | |
| 9 | | | 5 |
| 10 | | | |
| 11 | | | |
| 12 | | | |
| 13 | | | |
| 14 | | | |
| 15 | | | |

Figure 5.13: Uniform grid using sorting. [20]

#### 5.5.1.1 Results

Figure 5.14 shows the GPU computation time per iteration to simulate the behaviour of the ship in the first ice field which has 456 ice floes for all five different durations. "Variable Radius" is the computation time using the list of neighbours approach that have been discussed in section 5.1 and "Uniform Grid" is the computation time using the uniform grid approach.

Figure 5.15 shows the speed up of using the variable radius approach over the uniform grid approach in all five different cases.

As seen in Figures 5.14 and 5.15 it is clear that the uniform grid approach is slower than the variable radius approach. Therefore, the variable radius approach was used in this work.

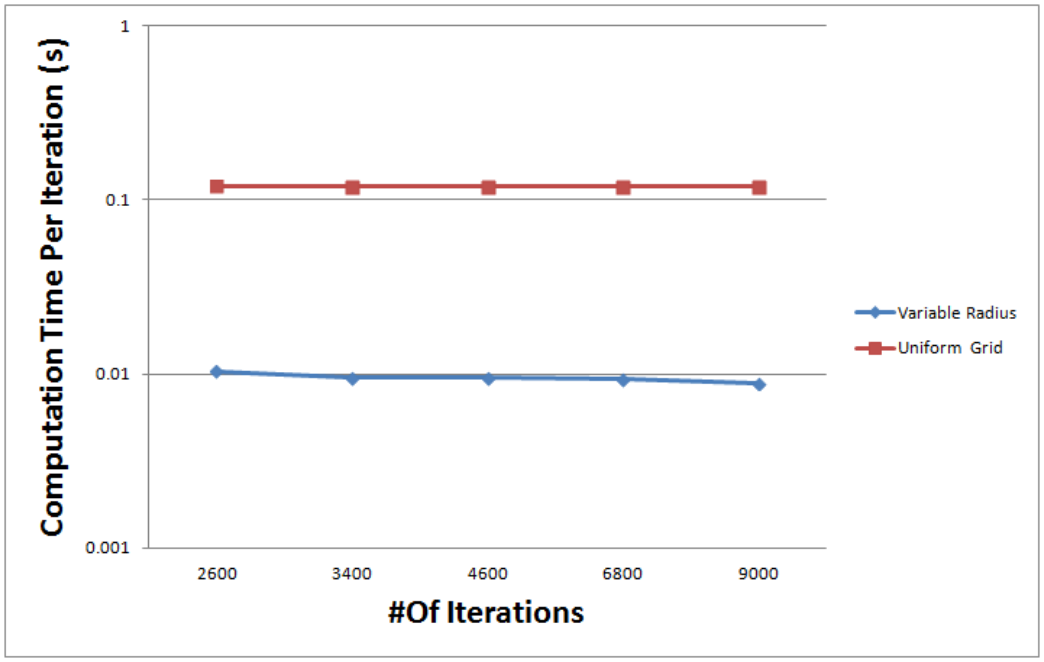



Figure 5.14: Computation time per iteration for the 456 ice field using the uniform grid and list of neighbours approaches.

### 5.5.2 Triangulation Mesh

The second alternative collision detection approach called triangulation mesh. Based on this approach to determine if two polygons intersects: each polygon is divided into triangles and then it is determined whether at least two triangles from the two polygons intersects.

#### 5.5.2.1 Results

Figure 5.16 shows the GPU computation time per iteration to simulate the behaviour of the ship in the first ice field which has 456 ice floes for all five different durations. “Variable Radius” is the computation time using the list of neighbours approach that

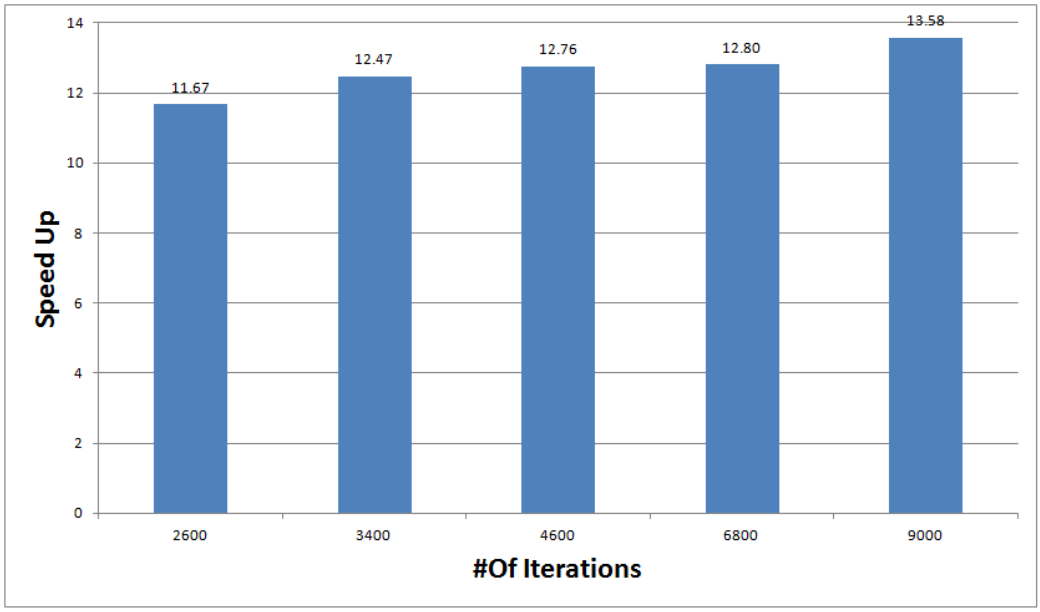



Figure 5.15: Variable radius approach speed up over uniform grid approach for the 456 ice field.

have been discussed in section 5.1 and "Triangulation Mesh" is the computation time using the triangulation mesh approach.

Figure 5.17 shows the speed up of using the variable radius approach over the triangulation mesh approach in all five different cases.

As seen in Figures 5.16 and 5.17 it is clear that the triangulation mesh approach is slower than the variable radius approach. Therefore, the variable radius approach was used in this work.

## 5.6 Kernel Block Size and Number of Blocks

As seen in Chapter 4, the kernel is executed as a grid of thread blocks. Therefore, the block size (number of threads per block) and the number of blocks per grid must

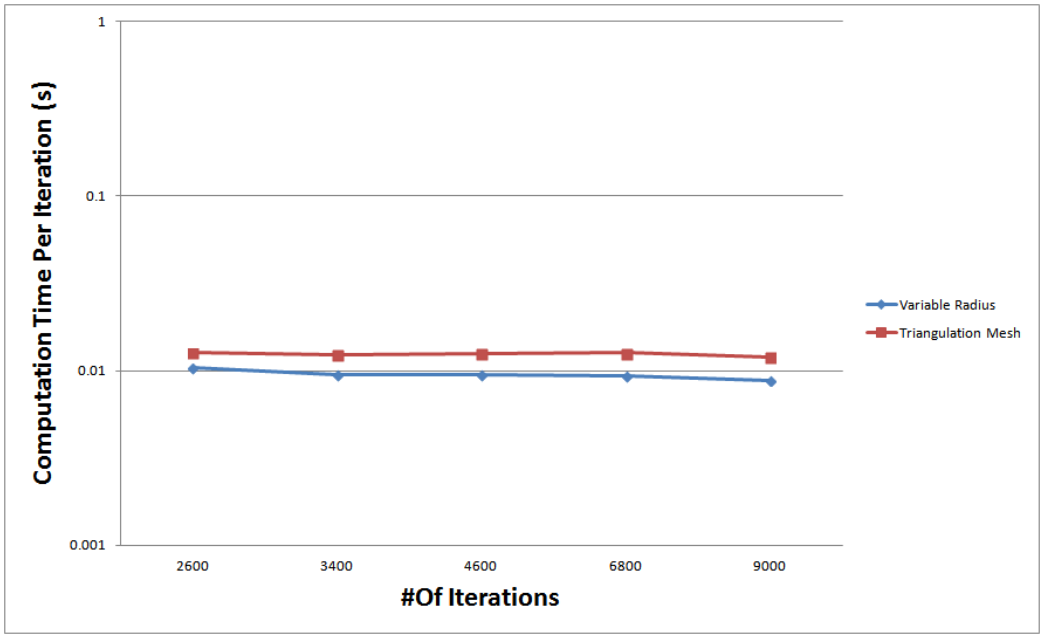



Figure 5.16: Computation time per iteration for the 456 ice field using the triangulation mesh and list of neighbours approaches.

be identified before the kernel is launched. A number of experiments were done to see the effect of the block size and number of blocks on the performance of the ice simulator. The simulator was run on two different ice fields for 3000 iterations with different block sizes (2, 4, 8, 16, 32, 64, 128, 256). The first ice field has 3584 ice floes and the second one has 7168 ice floes. Then, the computation time per iteration is measured, which shown in Figure 5.18.

As seen in Figure 5.18, the computation time per iteration is higher for smaller block sizes and when the block size is 32 and higher the computation time per iteration gets better. This is due to the number of multiprocessors on the GPU card that were used (14). Also, each multiprocessor has 32 cuda cores. Therefore, to get the full use of the multiprocessors the block size should be a multiple of the number of cuda

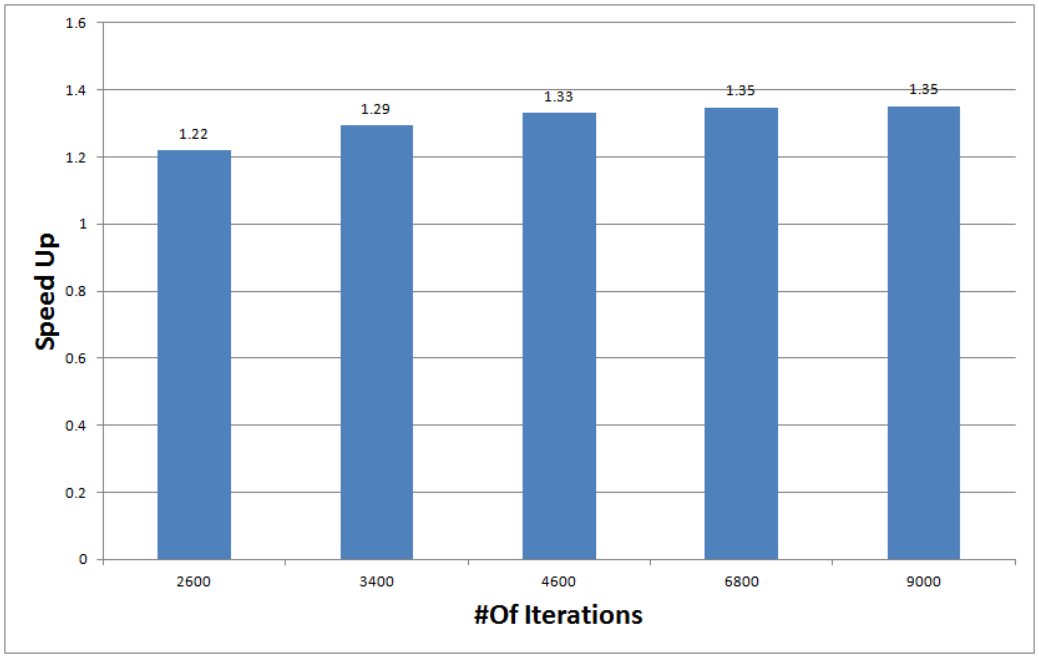



Figure 5.17: Variable radius approach speed up over triangulation mesh approach for the 456 ice field.

cores per multiprocessor on the GPU card (32) and the number of blocks should be a multiple of the number of multiprocessors on the GPU card (14). When the block size is 32, the number of blocks for the 3584 ice field is 112 (=3584/32). Since one thread is used for each ice floe, the number of blocks is equal to the number of ice floes divided by the block size. Tables 5.1 & 5.2 show the number of blocks for each ice field.

## 5.7 Data Transfer between the GPU and the CPU

It is known that the data transfer between the GPU and the CPU is a big bottleneck in achieving high performance in GPU/CPU applications. Therefore, to reduce the

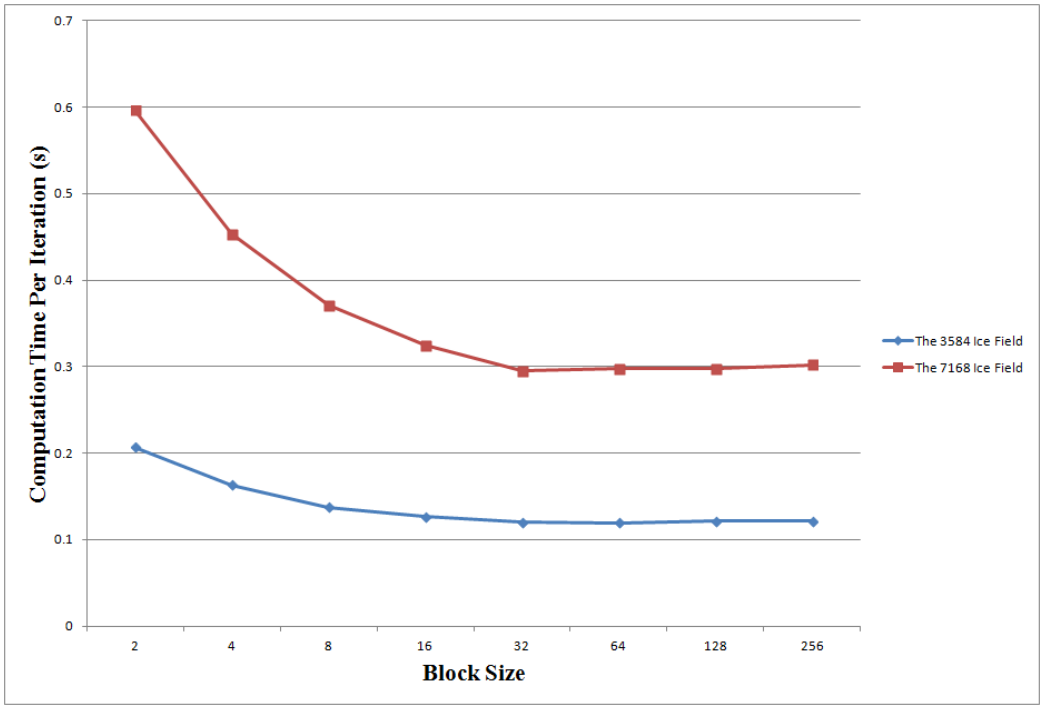



Figure 5.18: Computation time per iteration using different block sizes

latency of transferring the data back from the GPU to the CPU, a parameter ($n$) was added to control the number of time steps taken before copying the data back to the CPU.

Another experiment was conducted to see the effect of copying the data back from the GPU to the CPU on the performance of the simulator. The experiment consists of simulating an ice field that has a 7168 ice floes for 3000 iterations and measuring the computation time per iteration for a different values of $n$ (the number of time steps). As seen in Figure 5.19, the computation time per iteration decreases as the number of timesteps increases. When the number of timesteps increases, the amount of data that is copied back from the GPU to the CPU decreases and therefore the

Table 5.1: The number of blocks for the 3584 ice field

| #Of Blocks | Block Size |
|---|---|
| 1792 | 2 |
| 896 | 4 |
| 448 | 8 |
| 224 | 16 |
| 112 | 32 |
| 56 | 64 |
| 28 | 128 |
| 14 | 256 |

Table 5.2: The number of blocks for the 7168 ice field

| #Of Blocks | Block Size |
|---|---|
| 3584 | 2 |
| 1792 | 4 |
| 896 | 8 |
| 448 | 16 |
| 224 | 32 |
| 112 | 64 |
| 56 | 128 |
| 28 | 256 |

computation time decreases.

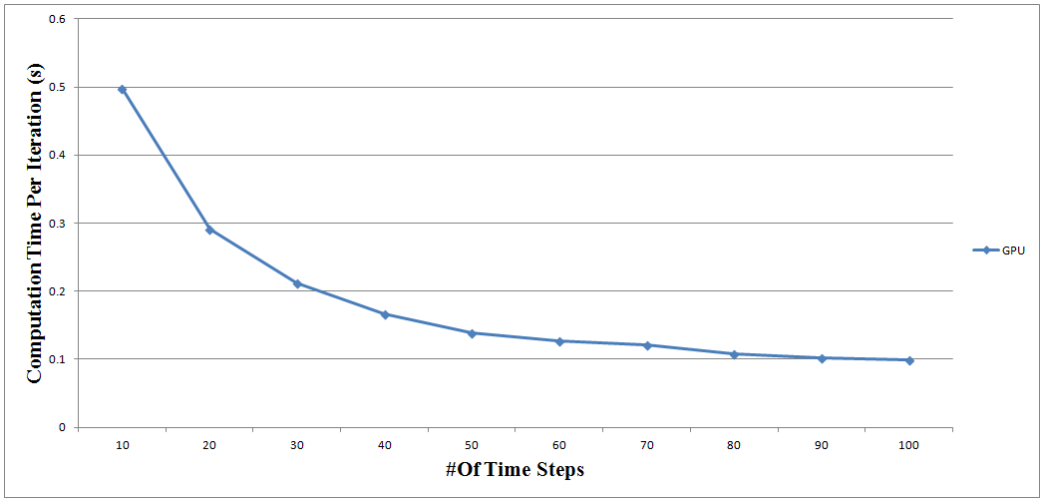



Figure 5.19: Computation time per iteration using different number of timesteps

# Chapter 6

# Model Validation and Applications

This chapter describes the experiments to validate the numerical model of ship operations in 2D pack ice. It also describes the useful applications that have been done using the GPGPU ice simulator in simulating and analysis vessel operations in pack ice.

## 6.1 Model Validation

A series of pilot experiments to validate the generic GPGPU model functionality and to identify points of improvement has been performed in cooperation with Roelf C. Dragt and was presented in a conference paper [16]. He was an exchange student in the STePS$^2$ project.

### 6.1.1 Modeling the GPGPU Model

The GPGPU model has been validated using physical model experiments. These experiments were designed to replicate the physical conditions of the GPGPU model as closely as possible. Most important were the degrees of freedom for the floes and the vessel. The floes have three degrees of freedom; movement in $x$-and $y$-direction and rotation around the $z$-axis. This means that rafting and rubbling are excluded. The ship is restricted to one degree of freedom, movement in $x$-direction (forward movement). Figure 6.1 shows the 2D concept and the axis used.

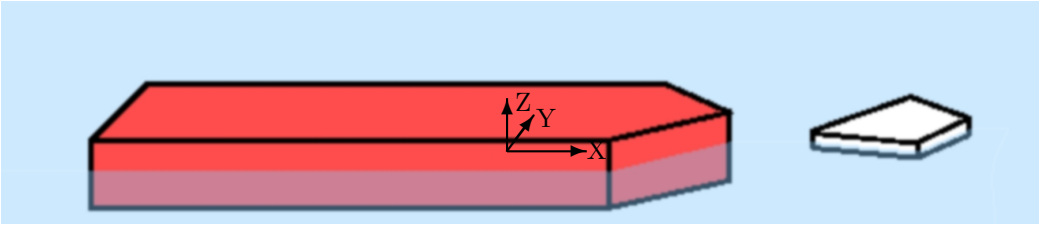



Figure 6.1: Schematic view of the 2D concept used in the model.

### 6.1.2 Model Experiments

The main goal is divided into three subgoals:

A To develop a repeatable method of creating ship-floe and floe-floe collisions in the lab that is consistent with the 2D formulation of the GPGPU model.

B To develop a method to compare the results to a numerical simulation.

C Validate the numerical model and make recommendations.

#### 6.1.2.1 Method of Creating Ship-Floe and Floe-Floe Collisions

The experiments are carried out in a transparent acrylic tank, located in the marine laboratory of Memorial University of Newfoundland's Engineering and Applied Sciences Faculty. The tank measures 7.9 meter in length, 1.47 meters wide and 0.97 meters deep and the walls are constructed out of acrylic glass to enable an all-round view, as is shown in Figure 6.2.

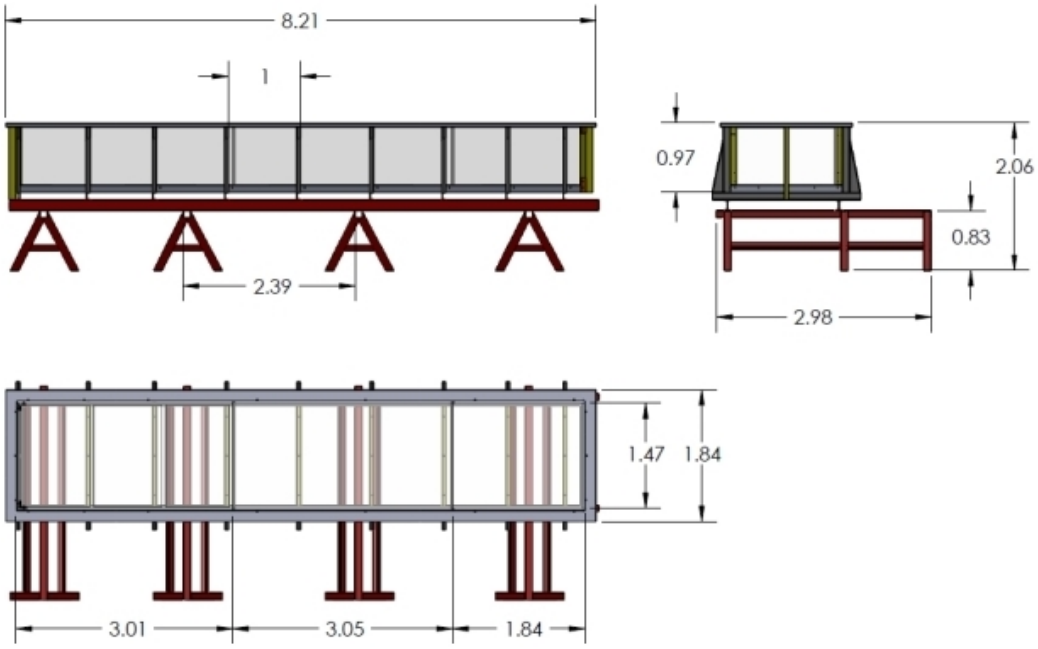



Figure 6.2: Drawing of the acrylic tank, dimensions are in meters.

The ship and the floes are constructed out of polypropylene, with a density of 905 kg/m$^3$, which closely approximates to the density of ice. Polypropylene was chosen because it has the right density and it doesn't melt. These properties are close to the conditions of the GPGPU model, which assumes rigid body behaviour. Finally, the material can be reused many times, which makes it an ideal material for tank testing.

The floes are 12.7 mm thick (1/2 inch) and randomly shaped into convex polygons with three to six sides. The vessel itself was made out of a 50.8 mm thick (2 inch) sheet of polypropylene, with an overall length of 0.91 m (36 inches) and a beam of 0.178 m (7 inches). A small recess was machined into the vessel to reduce the weight and increase the freeboard. The floes and the vessel do not change shape over the depth, because of the 2D restriction. Figure 6.3 shows the vessel with her main dimensions.

The vessel is controlled using an overhead carriage, which is connected to the vessel using an aluminium tow post. The carriage is suspended and moved by a wire loop, mounted over the centerline of the tank. The wire is driven by a variable speed motor, which is controlled by a DC controller (see Figure 6.4). Unfortunately, the overhead carriage was not stiff enough to restrict all the vessel's movements in sway and yaw direction. Therefore, the criteria set for the experiment (the vessel only moves in $x$-direction) was not entirely met. However, the error introduced is relatively small, as is shown in subsection 6.1.2.2.

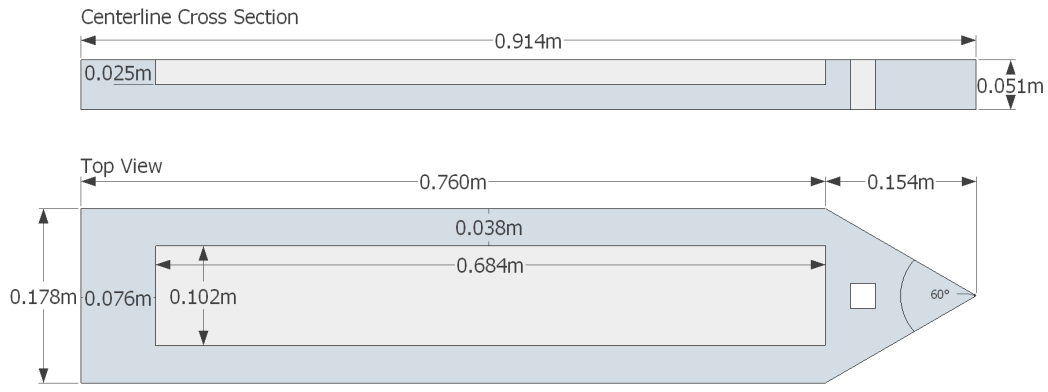



Figure 6.3: Design drawing of the vessel used in the experiments.

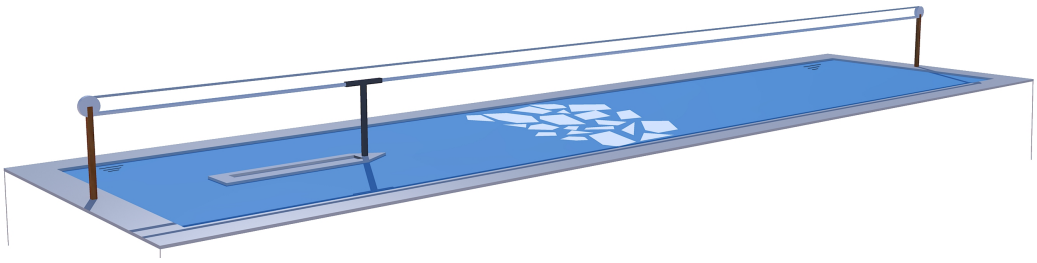
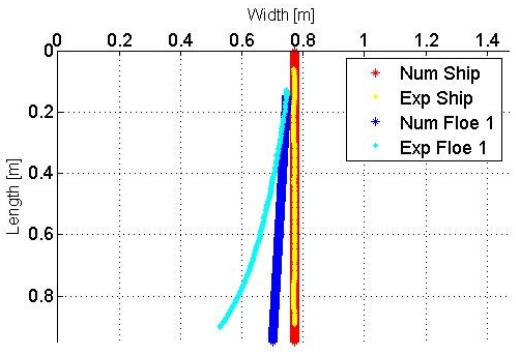
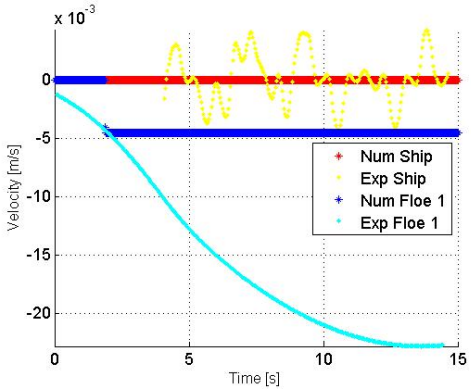


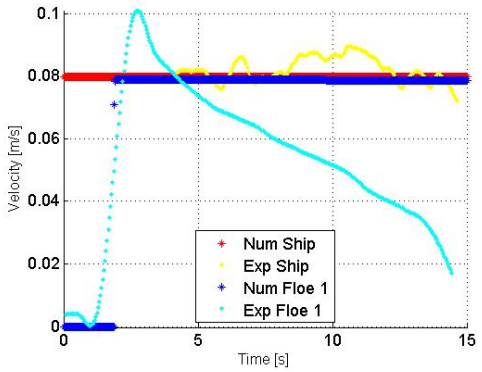
Figure 6.4: Schematic experiment layout, showing the vessel, some floes and the towing carriage above the tank.




(a) Position in 2D space

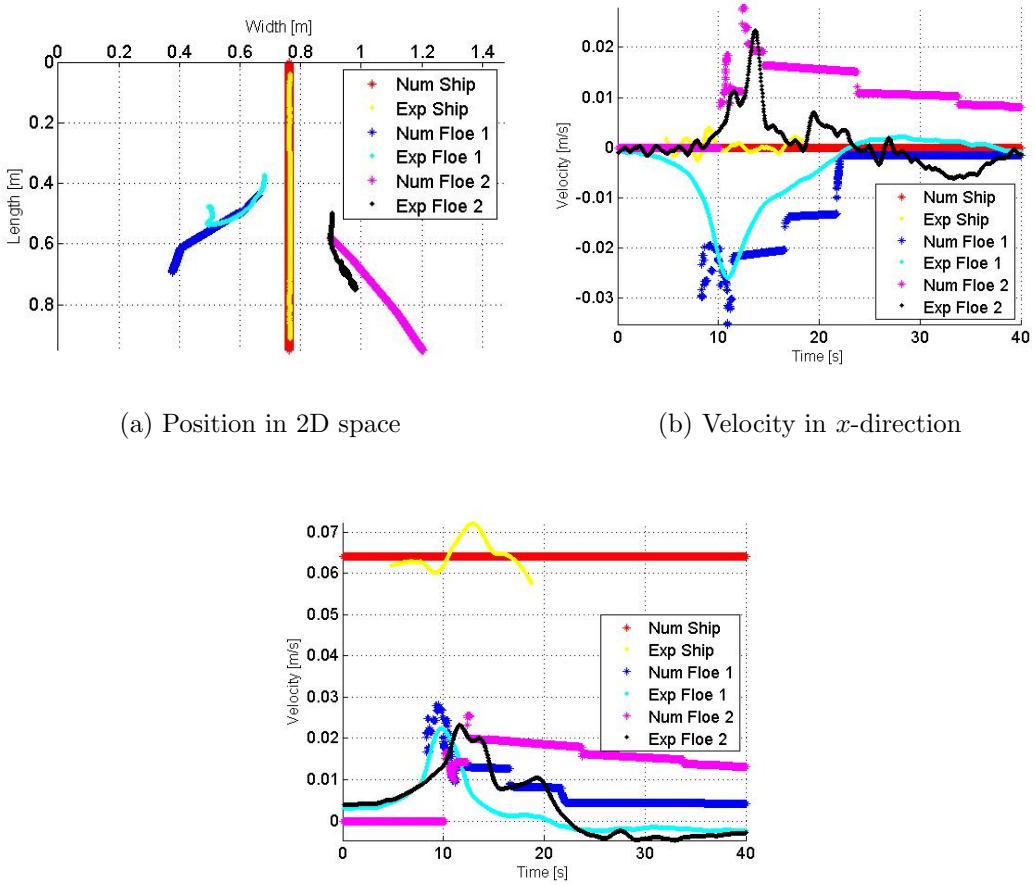



(b) Velocity in $x$-direction




(c) Velocity in $y$-direction

Figure 6.5: Comparison between the numerical model (Num) and experimental data (Exp) of a one ship and one floe situation.




(a) Position in 2D space

(b) Velocity in $x$-direction

(c) Velocity in $y$-direction

Figure 6.6: Pack ice comparison, numerical model (Num) and experimental data (Exp).

#### 6.1.2.2 Comparison of Experimental Data to Numerical Simulation

The starting position of all the floes, is manually converted into an .ice file. This file type is used as the input for the GPGPU simulation and contains all the positions and initial velocities of the bodies (vessel, floes and sides).

The GPGPU simulation processes the .ice input file and the resulting position

and velocities for each floe and the vessel over time are compared with those from the experiment, creating a plot with overlaying directions and velocity profiles. A situation with one floe and the vessel is shown in Figure 6.5 and a pack ice simulation is shown in Figure 6.6. Both figures display the position (a), velocity in $x$-direction (b) and velocity in $y$-direction (c). The experimental data contained some noise, which is filtered by averaging. Also, due to the resolution of the camera and the thresholding method, a change in centroid of just a couple of pixels induces in velocity.

Also, the ship in the experiment is able to sway a little, which is visible on the graphs. However, these disturbances are relatively small compared to the floe velocities.

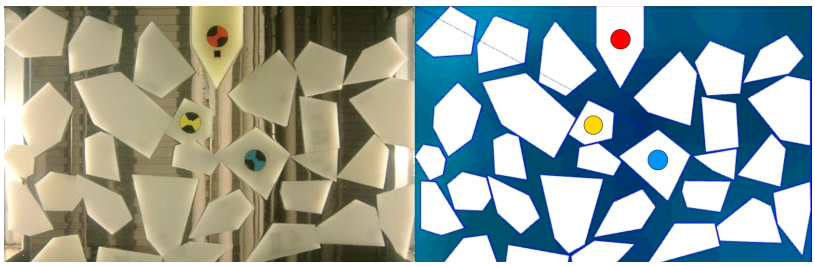
Finally, the graphical output of the numerical model enables qualitative comparison with the experimental data by placing both videos next to each other, as is shown for four frames in Figure 6.7.

#### 6.1.2.3 Numerical Model Validation and Recommendations

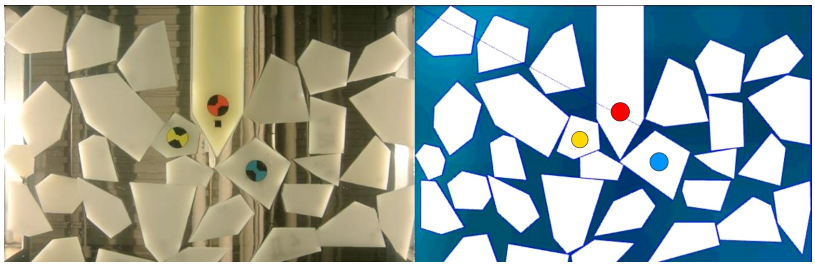
The model is validated in a qualitative way, visually comparing the data from the experiment with the GPGPU simulations. Conclusions can be drawn from this comparison, because the data sets are obviously different.

Based on the comparison of four experiments with only one floe and the vessel and one experiment with thirty floes and one vessel, the conclusions are as follows:

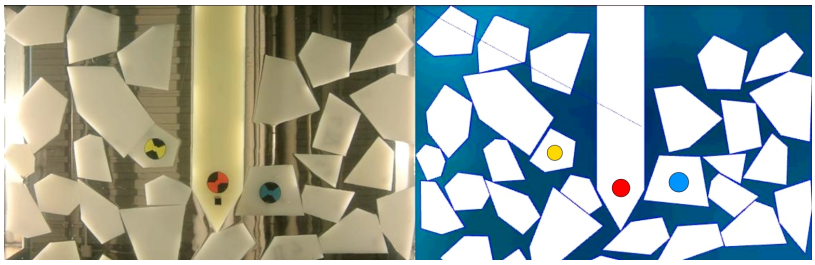
1. The hydrodynamics of the floes (water drag, added mass and wave damping) are likely is need of improvement in the GPGPU model. This shows floes (in



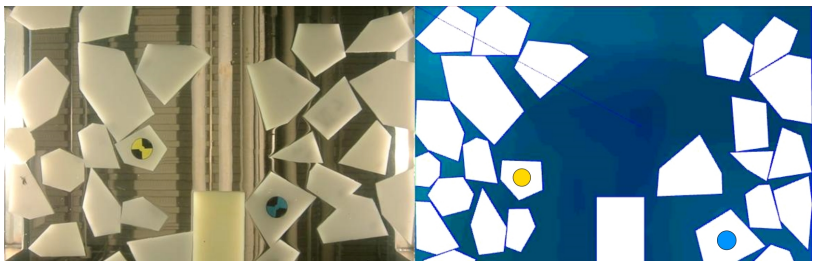
(a) t ≈ 4 sec



(b) t ≈ 8 sec



(c) t ≈ 12 sec



(d) t ≈ 25 sec

Figure 6.7: Comparison between the numerical simulation and the experiments of a single case. The bodies in the numerical model are shown with coloured dots for clarity.

open water, see Figure 6.5) loosing little velocity over time compared to the experiments. Since the model is used to model pack ice, open water behaviour is of less importance than collisional behaviour. However, it does influence the speed at which the floes collide and thus influences the "chain of events".

2. The GPGPU model, in pack ice situations (Figure 6.6), shows positions and velocities at the early stage of the simulation which are close to the experimental values. This leads to the conclusion that the collisions are modelled quite realistically. However, over time the average velocity of the floes in the numerical model is still higher than the velocity of the floes in the experiment, due to the low loss of energy in from hydrodynamics factors.

3. In the experiment, it is noticeable that the surface tension makes floes stick together, influencing their motions and speeds. It is clearly seen how the floes follow a different trajectory in Figure 6.6(a) and 6.7. This is not incorporated in the model (because in large scale, it is neglectable) but is important in the scale used for the experiments.

## 6.2 Applications

This section describes the applications that have been done using the GPGPU ice simulator in simulating and analysis vessel operations in pack ice. Which have previously been reported in IceTech [11] and OTC [10].

### 6.2.1 GPU Modeling of Ship Operations in Pack Ice

The work in this section has been done in cooperation with Dr. Claude Daley. A set of simulation domains, each containing hundreds of discrete and interacting ice floes is modeled. A simple vessel is modeled as it navigates through the domains. Each ship-ice collision is modeled, as is every ice-ice contact. Time histories of resistance, speed and position are presented along with the parametric sensitivities. The results are compared to published data from analytical, numerical and scale model tests.

The problem explored here is the transit of a vessel through open pack ice (see Figure 6.8), with floes ranging in size from 1m to 20m. A ship transiting this kind of ice cover will not only collide with many floes, but the ice floes will interact with each other in a complex way. A very large number of interactions will occur as a vessel travels even one ship length. The complexity of the problem is more readily handled by using the parallel computing power of a GPU.

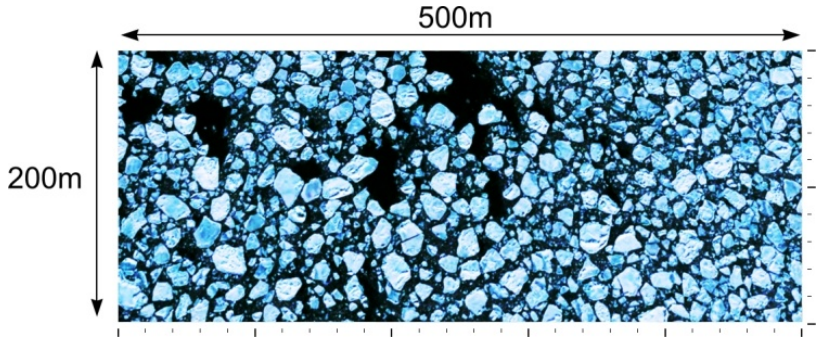



Figure 6.8: Example of natural first year pack ice. [11]

The simulation results given here represent only a first step in the use of this technology. The longer term aim of the project is to permit realistic and rapid simu-

lation of a wide range of ship-ice and ice-structure interactions and operations. The simulations presented in this work, involving simultaneous interactions of hundreds of ice floes have been performed at computational speeds up to 6x real time.

#### 6.2.1.1 Model Input

**6.2.1.1.1 Ice Conditions** The simulations presented below were performed in eight different ice fields. Six of the fields involved randomly shaped and oriented pack ice of varying concentration (see Figure 6.9 and Figure 6.10), while two involved regular arrays of equally sized hexagons (see Figure 6.11).

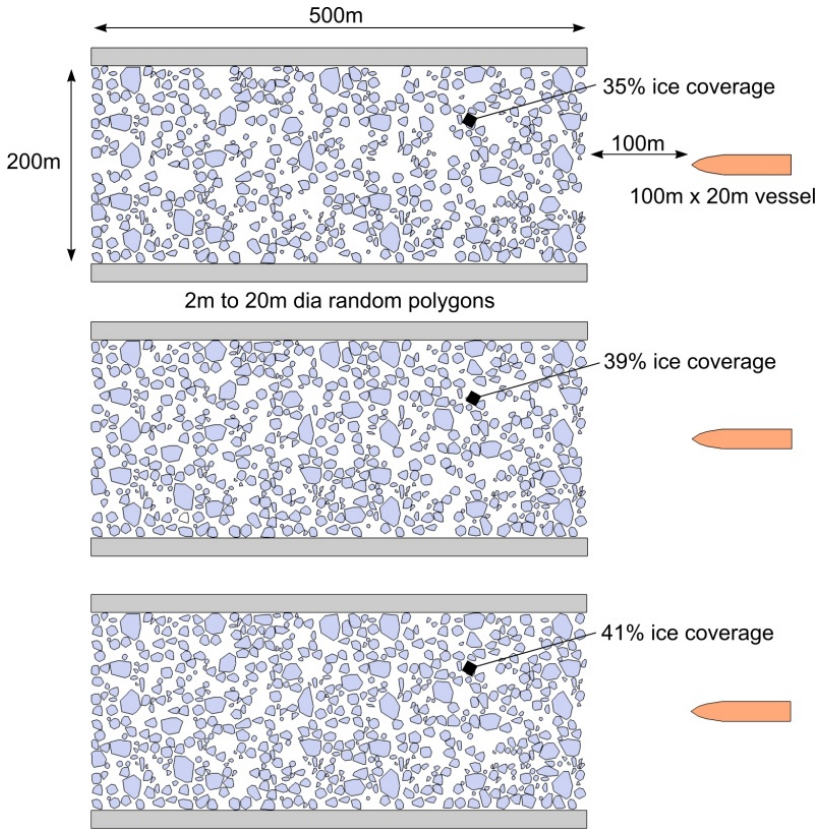



Figure 6.9: 35%, 39% and 41% simulation domains. [11]

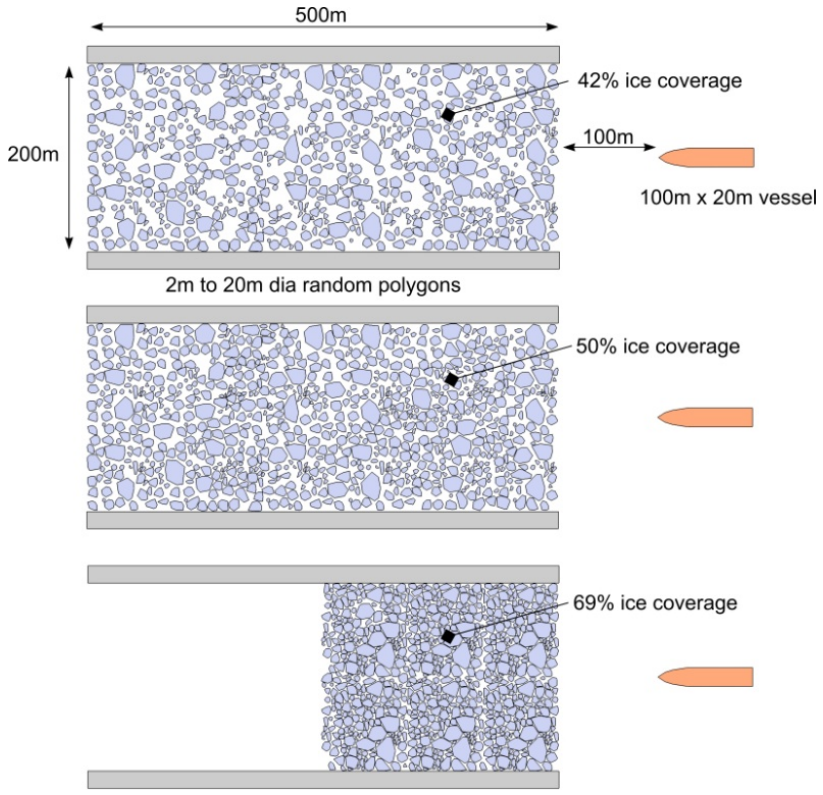



Figure 6.10: 42%, 50% and 69% simulation domains. [11]

For the random polygon cases, the ice floes were all represented as convex polygons of less than 20 sides. Floes were typically 4 to 7 sided (see Figure 6.12). The floe characteristic dimensions (defined as the square root of the area) ranged from 2m to 20 m, with a mean of 6.9m and a standard deviation of 3.9m. The floe set was created by drawing polygons of several of the floes in Figure 1 and then making multiple copies of the floes. The different concentrations were created manually by copying floes to increasingly fill in the gaps. For numerical reasons all the simulations started with no floes in contact with any other floes.

For the hexagonal polygon cases, the floes were all the same size, with a char-

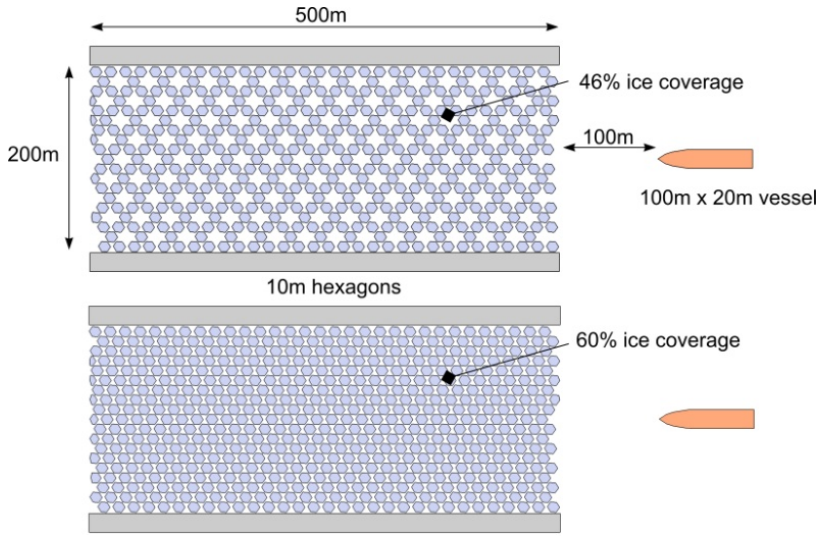



Figure 6.11: 46% and 60% simulation domains (hexagons). [11]

acteristic dimension of 10.1m (see Figure 6.13). The polygons were slightly rotated, with the intent of breaking the perfect symmetry and diminishing the tendency to interlock.

**6.2.1.1.2 Vessel Description** The vessel used in the simulation has the following nominal properties:

- Length: 100m
- Beam: 20m
- Mass: 7200 tonnes
- Geometry: 2D polygon (see Figure 6.14, 6.15)

Table 6.1: List of simulation run parameters.[11]

| Run #s | Number of Floes | Ice Coverage | Bollard Thrust [kN] | geometry |
|---|---|---|---|---|
| 1.1 to 1.5 | 560 | 35% | 23, 46, 92,178, 370 | random |
| 2.1 to 2.5 | 581 | 39% | 23, 46, 92,178, 370 | random |
| 3.1 to 3.5 | 618 | 41% | 23, 46, 92,178, 370 | random |
| 4.1 to 4.5 | 657 | 42% | 23, 46, 92,178, 370 | random |
| 5.1 to 5.5 | 456 | 46% | 23, 46, 92,178, 370 | hexagonal |
| 6.1 to 6.5 | 824 | 50% | 23, 46, 92,178, 370 | random |
| 7.1 to 7.5 | 595 | 60% | 23, 46, 92,178, 370 | hexagonal |
| 8.1 to 8.5 | 721 [a] | 69% | 23, 46, 92,178, 370 | random |

[a] in this case there field was 200x 250m instead of the normal 200x500m

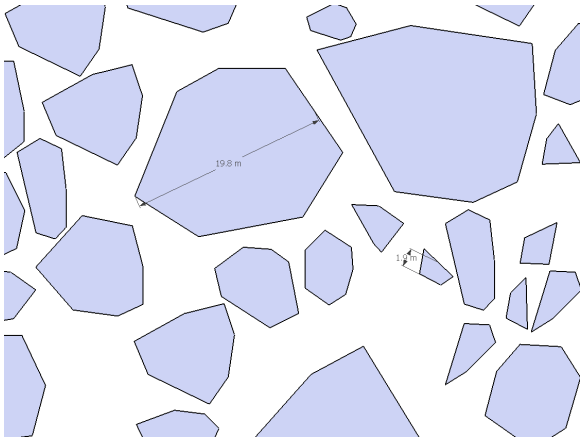



Figure 6.12: Close-up of random polygonal ice floes. [11]

#### 6.2.1.2 Model Mechanics

**6.2.1.2.1 Ice Behaviour** As stated above, the concept for the simulation is the rapid assessment of a sequence of discrete interactions with a large number of discrete ice objects. The transit of a vessel through pack ice, and the interactions of the ice are modeled as a set of contact events. The movements are treated using simple equations of motion. The individual ice blocks move in the 2D space of the simulation. The position and velocity of each floe is updated every time step. A simple water drag

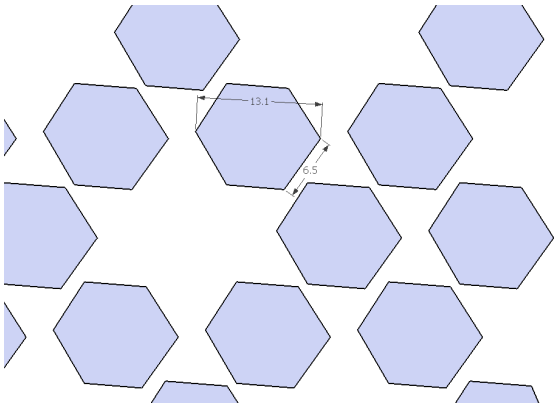



Figure 6.13: Close-up of hexagonal ice floes. [11]

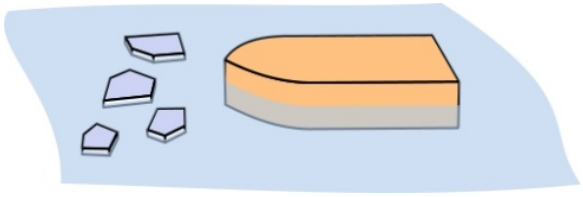


Figure 6.14: Sketch of 2D concept used in simulations. [11]

model results in the floes tending to slow. Ice-ice interactions account for both ice crushing impact forces and steady elastic stresses to resist static pressure. In this generation of the model there are no environmental driving forces (wind, current), nor are there any of the more complex responses such as rafting and rubbling. These are being planned for future generations of the model.

**6.2.1.2.2 Vessel Behaviour** The vessel is modeled as only moving forward with a simple self-propulsion algorithm. A simple water resistance model is combined with a simple thrust deduction model to produce a simple net-thrust vs. speed effect. In open water, the vessel will accelerate until the net thrust is zero, and then settle at its open water speed. In pack ice the sequence of ice forces will, on average, balance

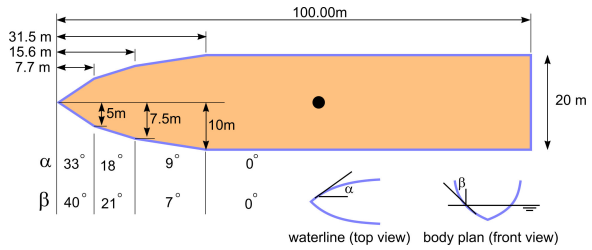



Figure 6.15: Geometry of vessel polygon. [11]

the available net thrust at some speed below the open water speed. In this way the average net thrust is a surrogate for time-averaged ice resistance. The process is not steady. Future versions of the model will include more aspects of vessel behaviour.

#### 6.2.1.3 Model Results

**6.2.1.3.1 Field Images** Figure 6.16 shows an image of a simulation taken as the vessel transits open pack ice. The vessel leaves a track of relatively open water along with a zone where the ice is more closely packed. The ice ahead and to the sides is undisturbed. A very large number is ship-ice and ice-ice contacts have taken place. Figure 6.16 shows a similar situation, but with 3 images overlaid using partial transparency. This makes it easier to see the ice floe disturbance (termed the "action zone"). The size and shape of the action zone changes as the ice cover becomes more concentrated.

**6.2.1.3.2 Time Sequence Results** Shown below are three time series plots for the simulation in 35% ice cover with a bollard thrust (it is a force that drives the ship) of 370kN. As the vessel moves through the ice, a sequence on impulses acts on the ship. The net thrust model tends to keep the ship moving and the vessel tends

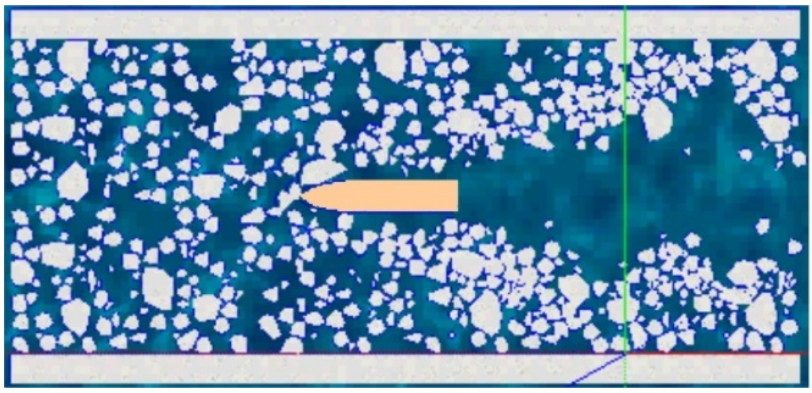


Figure 6.16: Image from simulation video in 35% coverage. [11]

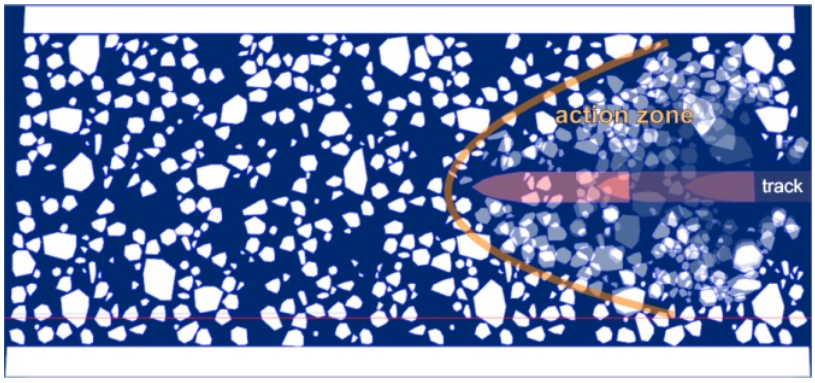



Figure 6.17: Image from simulation video in 35% coverage showing action zone. [11]

to settle down to a speed where the ice forces balance the available net thrust. The process is not steady because the ice forces are a series of very short impulses mixed with relatively long periods of no ice loads. Figure 6.18 shows a portion of the ice impact forces on the vessel. The ice forces are very quick, but do tend to last longer than one simulation step due to the number of floes in contact and the turning (and thus re-impact) of the floes. If the entire time history of this data were shown, it appears to be just a sequence of spikes.

Figure 6.19 shows the vessel speed for the entire simulation. At the start, the

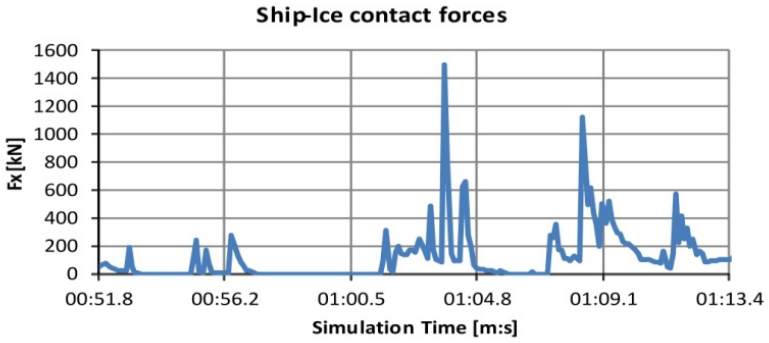



Figure 6.18: Partial time-history of ice collision forces on the vessel 35% coverage. [11]

vessel is set moving at its open water speed. As it enters the ice field it quickly slows to a nearly steady ice speed, though still with fluctuations. The fluctuations are due to the ice impulse loads. Figure 6.20 shows the net thrust. This time-averaged value of net thrust is effectively the ice resistance, as long as the net acceleration is close to zero.

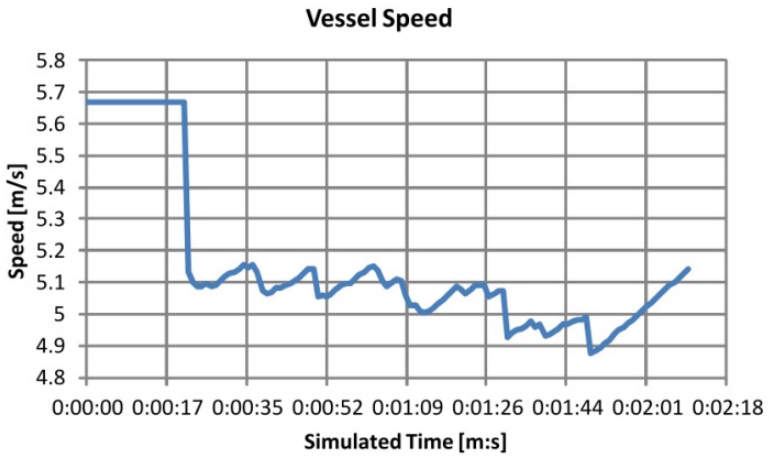



Figure 6.19: Vessel speed during simulation 35% coverage. [11]

These plots are representative of the simulations performed. Each impact is

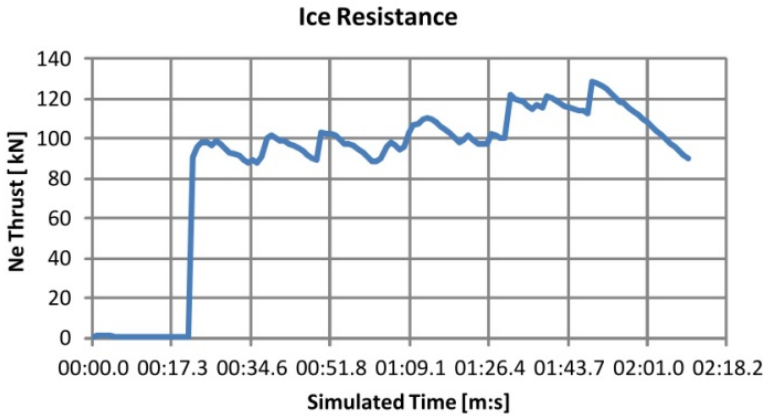



Figure 6.20: Net thrust during simulation in 35% coverage. [11]

tracked. Considerably more data is available for extraction from the simulations, such as the exact location of the impact on the hull. The approach also lends itself to easily including stochastic distributions of ice geometric and strength properties (shape, thickness, strength), which would generate additional data for parametric relationships.

**6.2.1.3.3 Parametric Results** To illustrate the general validity of the approach as well as to identify areas for improvement, the following section presents parametric trends in the results. The influence of velocity and ice concentration will be presented and compared to other published data. In the plots below (Figure 6.21 to Figure 6.27) the data labelled GPU refers to the present results. WC(2010) refers to an empirical model based on physical model tests [63]. MA(1989) refers to an analytical model of resistance in pack ice [42].

Ice resistance vs. velocity for various ice concentrations is given in Figure 6.21 to

Figure 6.25. The plots show two noteworthy aspects. The agreement with MA(1989) is remarkably good, while the agreement with WC(2010) is much less so. This is likely due to several reasons. The MA(1989) model made essentially the same assumptions about contact and energy that are in the GPU simulation. In both cases, all collisions are inelastic, such that energy is absorbed in ice crushing and water drag while momentum is conserved.

The WC(2010) model has a quite different basis. For one thing the WC model is an empirical fit to model test data at higher concentrations and much lower relative speeds. This means that there is some potential for error in the extrapolation to lower concentrations and or the higher speeds of this study. Secondly and more importantly, the WC physical tests contained a number of physical behaviours that were not part of the GPU model. In the physical model tests the ice was able to flex, raft, and rubble, as well as submerge below a 3D ship shape. These additional behaviours would result in different trend. There is also the likelihood that the ice sizes and shapes were different, which may have made a difference. As evidence of this, the GPU simulations in the 60% regular hexagonal pack ice resulted in noticeably higher resistance than in random floes. This appeared to be the result of mechanical interlocking among the floes.

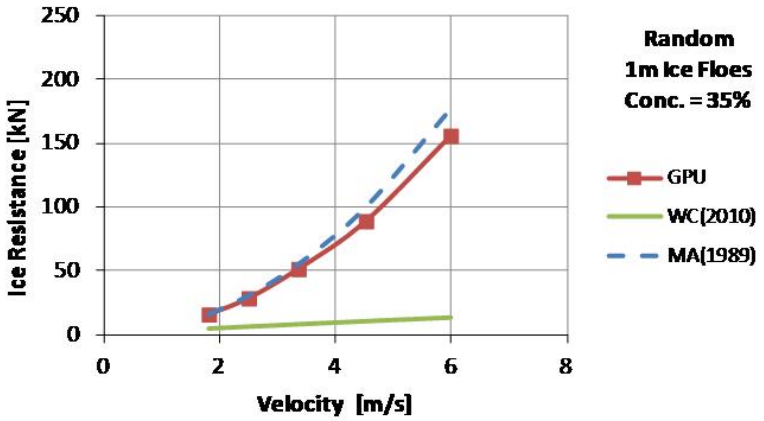



Figure 6.21: Comparison of resistance estimates in 35% coverage. [11]

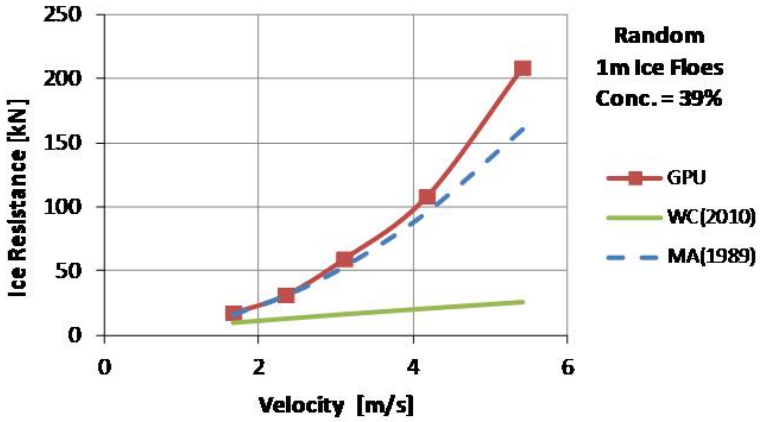



Figure 6.22: Comparison of resistance estimates in 39% coverage. [11]

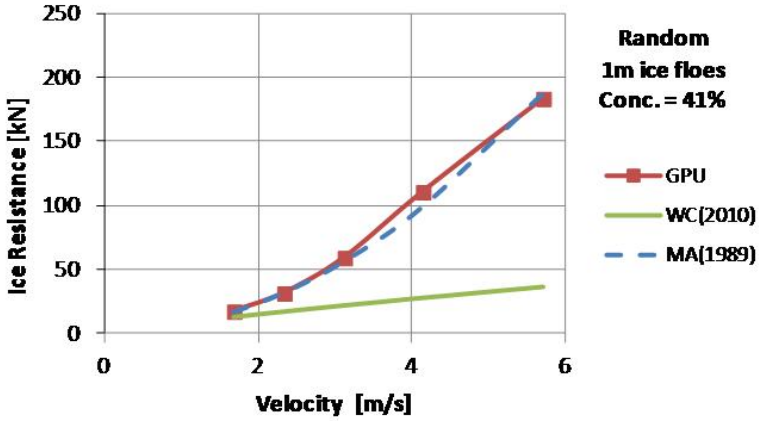



Figure 6.23: Comparison of resistance estimates in 41% coverage. [11]

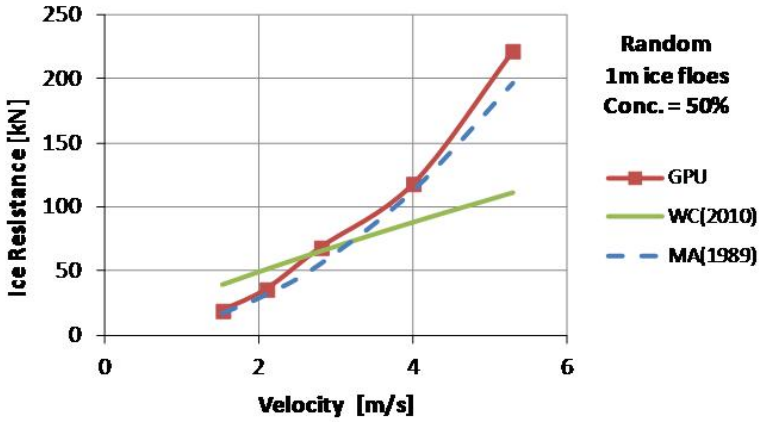



Figure 6.24: Comparison of resistance estimates in 50% coverage. [11]

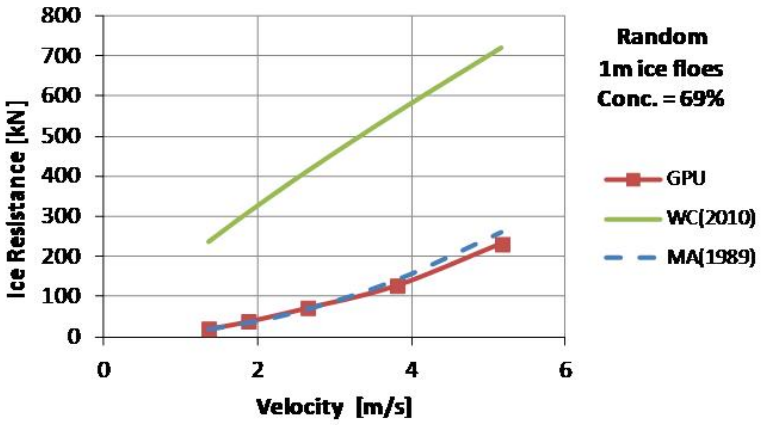



Figure 6.25: Comparison of resistance estimates in 69% coverage. [11]

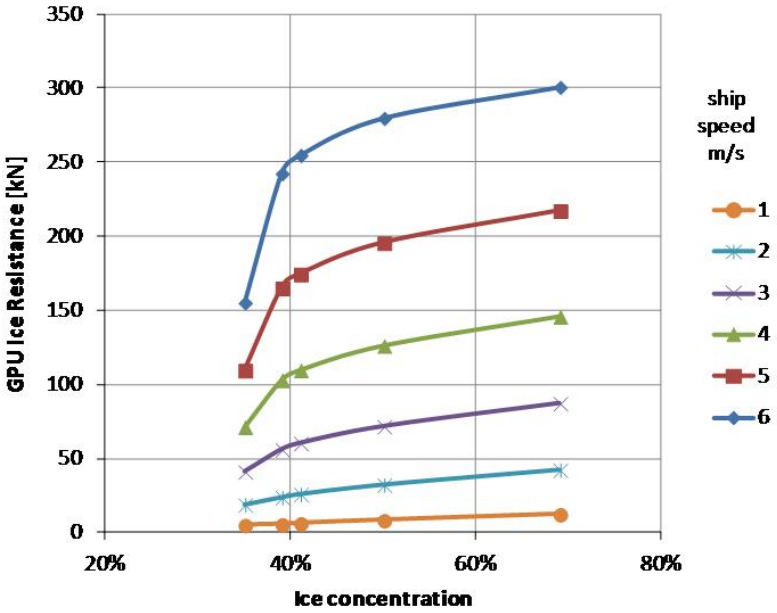



Figure 6.26: GPU model resistance estimates vs. velocity. [11]

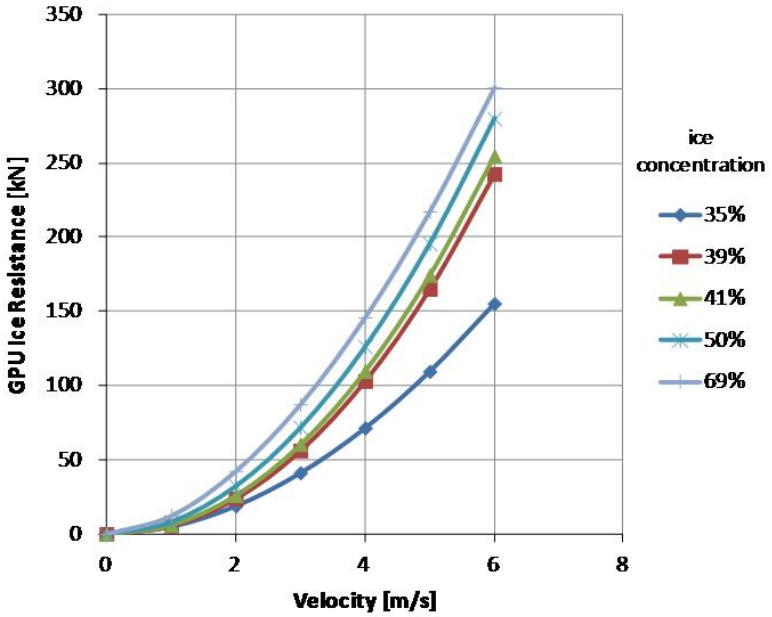



Figure 6.27: GPU model resistance estimates vs. concentration. [11]

Figure 6.26 shows the trends of resistance vs. velocity for all the concentrations with random floes. The curves are approximately quadratic (i.e. exponent on velocity is close to two).

Figure 6.27 shows the trends vs. ice concentration. One interesting aspect to note is that the relationship is close to linear at slower speeds and becomes much less so at higher speed. This could be the result of the change in the size of the action zone as speed increases.

### 6.2.2 Ice-Event-Mechanics Evaluation of Ice Impact Load Statistics

The work in this section has been done in cooperation with Dr. Claude Daley. In this work the use of a Ice-Event-Mechanics simulation to assess local ice loads on a vessel operating in pack ice has been explored. A simple vessel is modeled as it navigates through the domain. Each ship-ice collision is modeled, as is every ice-ice contact. Each ship-ice collision event is logged, along with all relevant ice and ship data. Thousands of collisions are logged as the vessel transits many tens of kilometres of ice pack. The resulting impact load statistics are qualitatively evaluated and compared to published field data. The analysis provides insight into the nature of loads in pack ice.

Ice class vessels are unique in a number of ways in comparison to non-ice class vessels. Hull strength, power, hull form and winterization aspects are all issues that raise special challenges in the design of ice class ships. This work focuses on matters of local ice loads which pertain to hull strength in ice class vessels. More specifically, the work examines the parametric causes of local ice loads and statistics that result as a ship transits through open pack ice.

The issue of pack ice transit is of interest to those wishing to operate safely in such conditions. One key question is that of safe operational speeds. Consider the special case of open pack ice, where floes are relatively small, numerous and resting in calm water. A vessel moving through such an ice cover would experience a series of discrete collisions. As long as a vessel moved very slowly, the loads would be very low. In

such a case the vessel could make safe and steady progress, even if it had a relatively low ice class. However, if the vessel attempted to operate more aggressively, impact speeds would increase and a higher ice class would be needed for safe operations. The investigation below provides some insight into the factors that influence the loads in this situation. These factors include hull form, speed, floe size and concentration, ice thickness, strength and edge shape. Most prior studies have tended to focus on ice thickness and strength as the primary determinants of load. This study shows that ice edge shape and mass, along with hull form and locations are also strong determinants of loads, and especially the load statistics. The simulations provide some interesting data, especially when compared to field trials data.

A related focus for the study is to explore the use of GEM approach. The GEM approach represents the integration of a number of concepts. The physical space is described as a set of bodies. The movement (kinematics) of the bodies is tracked using simple equations of motion. Time is divided into relatively long 'moments', during which events occur. All variables in the simulation—forces, movements, fractures and other changes—are considered to be aspects of events. Some events are momentary, while others are continuing. Some events involve a single body and are termed solo events. Motion, for example, is treated as a solo event. Some events are two-body events. Impact is an example of a two-body event. The GEM approach lends itself to parallel implementation, which in this case is accomplished in a GPU environment.

The event models are the analytical solutions of specific scenarios. As a result, the events do not require solution (in the numerical sense) during the GEM simulation.

The event solution is merely invoked for the specific inputs that arise at that point in the GEM simulation. For example, the collision load depends on the specific shape and position of the ice floe, as well as thickness, flexural strength and crushing behaviour. The load also depends on hull form and impact location, as well as the mass properties of the ship. There are dozens of input variables which influence the specific event parameters. Nevertheless, the computation problem is far smaller than if the continuum mechanics were to be solved for each collision event. The GEM model focuses on the large scale system involving a large number of bodies, rather than on any single impact.

#### 6.2.2.1 Impact Algorithm Check

The collision model used in the GEM simulation has a relativly simple analytical solution that can be solved in a spreadsheet. To check that the GEM software is producing the expected impact results for a variety of cases, a set of 32 calibration impacts (See Figure 6.28) were modeled in both the GEM program and a spreadsheet. In each of the 32 cases a 10m x 10m ice floe was placed directly in front of the vessel and allowed to strike. The GEM forces were compared to the spreadsheet results. The comparison is shown in Figure 6.29. There were some small differences attributed to the slight differences in the contact locations that arise in the numerical model. Overall the agreement is excellent and confirms that no gross errors occurred in the implementation

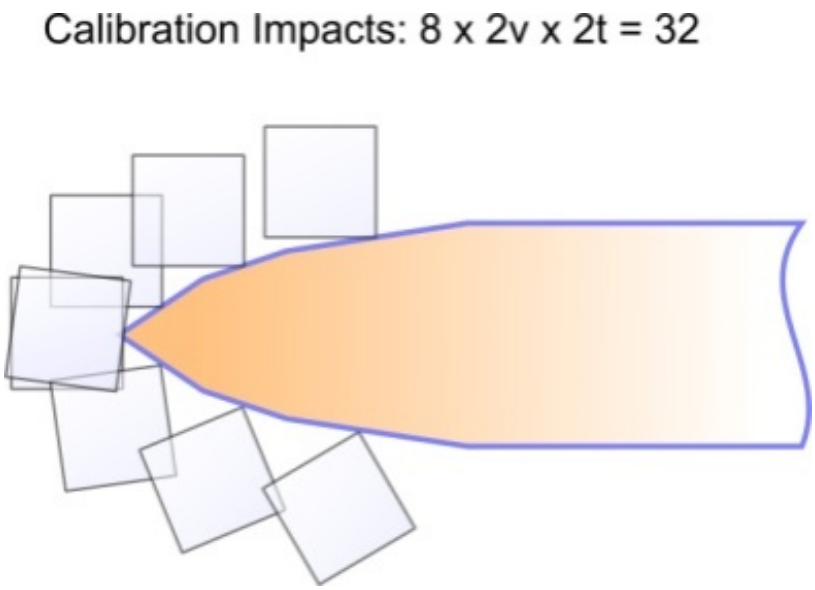



Figure 6.28: Calibration impact Cases. [10]

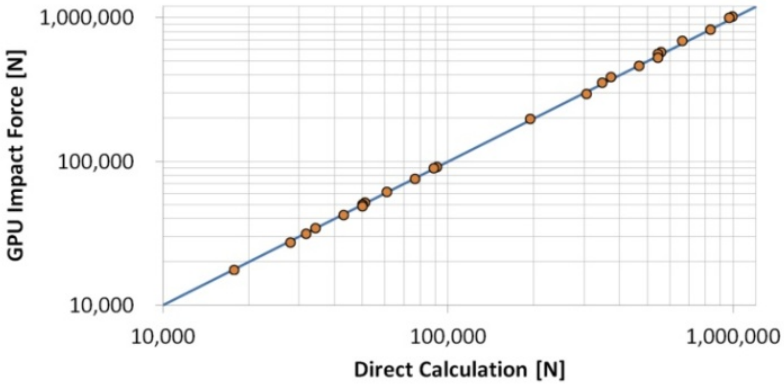



Figure 6.29: Direct vs GEM impacts compared for validation. purposes[10]

#### 6.2.2.2 Simulation Description

The simulations presented all involve a ship transiting through a 200m x 500m pack ice region at a set power level. One example case is shown in Figure 6.30. The ice represents 4/10th ice cover, with a mix of thin, medium and thick first year ice (0.5m,

0.7m and 1.2m floes). The floes are random in size (same range for each thickness). The egg code (The basic data concerning concentrations, stages of development (age) and form (floe size) of ice are contained in a simple oval form. A maximum of three ice types is described within the oval) that represents the ice is also shown in Figure 6.30. Tables 6.2,6.3 describe the 70 individual runs that form the data for this paper. A summary of the key simulation parameters and results are given. The ice floes are comprised of 3 groups of random polygons. Each group can be assigned a common thickness and in this way a wide variety of cases can be developed depending on which thickness values are assigned to which ice group. Two of the groups represents 1/10th coverage (10% of the surface area) while one group represents 2/10th coverage. In total there are 668 unique ice floes, which are combined in various ways and assigned various thicknesses in the various runs.

In total, in the 70 runs performed there were 28,685 ship-ice collisions recorded, which are the basis of the analysis presented. It should be noted that many more ice-ice and ice-wall collisions were simulated but were not logged, nor were the ice resistance values. The GEM approach lends itself to a variety of potential uses.

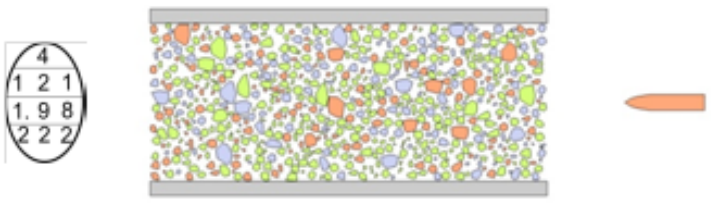



Figure 6.30: Ice Conditions for runs 46 through 50.[10]

Table 6.2: Listing of first 35 run cases, with summary result values.[10]

| Run No. | Run Name* | Thrust level [kN] | 1.2m/0.7m/0.5 m ice coverage % | No. of Floes | Floe area mean / SD [m2] | No. of Impacts | Ship speed m/m/m** [m/s] | Impact Force m/m/m ** [kN] |
|---|---|---|---|---|---|---|---|---|
| 1 | 001_46 | 46 | 0 / 0 / 10 | 159 | 63 / 70 | 49 | .94/1.5/1.9 | 0/64/529 |
| 2 | 001_92 | 92 | 0 / 0 / 10 | 159 | 63 / 70 | 50 | 1.3/2.0/2.6 | 1/94/588 |
| 3 | 001_185 | 185 | 0 / 0 / 10 | 159 | 63 / 70 | 58 | 1.9/2.8/3.6 | 0/120/651 |
| 4 | 001_370 | 370 | 0 / 0 / 10 | 159 | 63 / 70 | 66 | 2.6/3.9/4.9 | 0/143/718 |
| 5 | 001_740 | 740 | 0 / 0 / 10 | 159 | 63 / 70 | 75 | 3.7/5.3/6.5 | 0/173/850 |
| 6 | 020_46 | 46 | 0 / 20 / 0 | 354 | 56 / 51 | 130 | .89/1.5/1.9 | 0/47/509 |
| 7 | 020_92 | 92 | 0 / 20 / 0 | 354 | 56 / 51 | 151 | 1.3/2.1/2.6 | 0/62/805 |
| 8 | 020_185 | 185 | 0 / 20 / 0 | 354 | 56 / 51 | 132 | 2.5/3.0/3.5 | 0/111/980 |
| 9 | 020_370 | 370 | 0 / 20 / 0 | 354 | 56 / 51 | 142 | 2.6/3.9/4.8 | 0/151/1097 |
| 10 | 020_740 | 740 | 0 / 20 / 0 | 354 | 56 / 51 | 171 | 3.5/5.4/6.4 | 0/176/1346 |
| 11 | 100_46 | 46 | 10 / 0 / 0 | 155 | 64 / 71 | 38 | 1.1/1.6/1.9 | 0/73/508 |
| 12 | 100_92 | 92 | 10 / 0 / 0 | 155 | 64 / 71 | 43 | 1.3/2.2/2.6 | 0/137/716 |
| 13 | 100_185 | 185 | 10 / 0 / 0 | 155 | 64 / 71 | 64 | 1.8/3.0/3.6 | 0/152/968 |
| 14 | 100_370 | 370 | 10 / 0 / 0 | 155 | 64 / 71 | 61 | 2.6/4.1/4.9 | 3/264/1601 |
| 15 | 100_740 | 740 | 10 / 0 / 0 | 155 | 64 / 71 | 64 | 3.6/5.6/6.4 | 0/381/2100 |
| 16 | 004_46 | 46 | 0 / 0 / 40 | 668 | 60 / 61 | 438 | 0.9/1.5/1.9 | 0/20/360 |
| 17 | 004_92 | 92 | 0 / 0 / 40 | 668 | 60 / 61 | 481 | 1.3/2.1/2.5 | 0/27/542 |
| 18 | 004_185 | 185 | 0 / 0 / 40 | 668 | 60 / 61 | 450 | 1.8/2.9/3.5 | 0/45/532 |
| 19 | 004_370 | 370 | 0 / 0 / 40 | 668 | 60 / 61 | 462 | 2.5/3.9/4.9 | 0/56/737 |
| 20 | 004_740 | 740 | 0 / 0 / 40 | 668 | 60 / 61 | 480 | 3.5/5.3/6.2 | 0/78/1152 |
| 21 | 040_46 | 46 | 0 / 40 / 0 | 668 | 60 / 61 | 470 | 0.9/1.5/1.8 | 0/25/445 |
| 22 | 040_92 | 92 | 0 / 40 / 0 | 668 | 60 / 61 | 449 | 1.3/2.0/2.5 | 0/34/753 |
| 23 | 040_185 | 185 | 0 / 40 / 0 | 668 | 60 / 61 | 477 | 1.8/2.8/3.4 | 0/53/1069 |
| 24 | 040_370 | 370 | 0 / 40 / 0 | 668 | 60 / 61 | 534 | 2.6/3.9/4.5 | 0/79/1194 |
| 25 | 040_740 | 740 | 0 / 40 / 0 | 668 | 60 / 61 | 476 | 3.5/5.3/6.1 | 0/107/1185 |
| 26 | 400_46 | 46 | 40 / 0 / 0 | 668 | 60 / 61 | 517 | 0.9/1.4/1.7 | 0/29/903 |
| 27 | 400_92 | 92 | 40 / 0 / 0 | 668 | 60 / 61 | 538 | 1.2/2.0/2.3 | 0/42/1011 |
| 28 | 400_185 | 185 | 40 / 0 / 0 | 668 | 60 / 61 | 499 | 1.8/2.7/3.2 | 0/66/1650 |
| 29 | 400_370 | 370 | 40 / 0 / 0 | 668 | 60 / 61 | 532 | 2.5/3.7/4.2 | 0/94/1823 |
| 30 | 400_740 | 740 | 40 / 0 / 0 | 668 | 60 / 61 | 474 | 3.5/5.0/5.7 | 0/146/2171 |
| 31 | 022_46 | 46 | 0 / 20 / 20 | 668 | 60 / 61 | 445 | 0.9/1.5/1.8 | 0/21/473 |
| 32 | 022_92 | 92 | 0 / 20 / 20 | 668 | 60 / 61 | 461 | 1.3/2.1/2.5 | 0/33/737 |
| 33 | 022_185 | 185 | 0 / 20 / 20 | 668 | 60 / 61 | 492 | 1.8/2.8/3.5 | 0/46/962 |
| 34 | 022_370 | 370 | 0 / 20 / 20 | 668 | 60 / 61 | 482 | 2.5/3.9/4.8 | 0/68/1080 |
| 35 | 022_740 | 740 | 0 / 20 / 20 | 668 | 60 / 61 | 417 | 3.5/5.3/6.2 | 0/103/1197 |

*Each run name is of the form: nnn_p. The first 3 'n's refer to the coverages in 10ths of the 3 different thicknesses.
The p number is the thrust level in KN. All runs were a transit of a 500 m long ice field ( x 200 m wide).
** min, mean and max values

Table 6.3: Listing of last 35 run cases, with summary result values.[10]

| Run No. | Run Name* | Thrust level [kN] | 1.2m/0.7m/0.5 m ice coverage % | No. of Floes | Floe area mean / SD [m2] | No. of Impacts | Ship speed m/m/m** [m/s] | Impact Force m/m/m ** [kN] |
|---|---|---|---|---|---|---|---|---|
| 36 | 031_46 | 46 | 0 / 30 / 10 | 668 | 60 / 61 | 478 | 0.9/1.4/1.8 | 0/17/904 |
| 37 | 031_92 | 92 | 0 / 30 / 10 | 668 | 60 / 61 | 471 | 1.3/2.1/2.5 | 0/32/835 |
| 38 | 031_185 | 185 | 0 / 30 / 10 | 668 | 60 / 61 | 445 | 1.8/2.8/3.4 | 0/57/1768 |
| 39 | 031_370 | 370 | 0 / 30 / 10 | 668 | 60 / 61 | 531 | 2.5/3.9/4.7 | 0/60/1073 |
| 40 | 031_740 | 740 | 0 / 30 / 10 | 668 | 60 / 61 | 463 | 3.5/5.3/6.1 | 0/100/1186 |
| 41 | 112_46 | 46 | 10/10/2020 | 668 | 60 / 61 | 523 | 0.9/1.4/1.8 | 0/22/719 |
| 42 | 112_92 | 92 | 10/10/2020 | 668 | 60 / 61 | 528 | 1.3/2.0/2.5 | 0/32/975 |
| 43 | 112_185 | 185 | 10/10/2020 | 668 | 60 / 61 | 612 | 1.8/2.8/3.3 | 0/42/1477 |
| 44 | 112_370 | 370 | 10/10/2020 | 668 | 60 / 61 | 569 | 2.5/3.8/4.6 | 0/57/1191 |
| 45 | 112_740 | 740 | 10/10/2020 | 668 | 60 / 61 | 496 | 3.5/5.3/6.1 | 0/106/1318 |
| 46 | 121_46 | 46 | 10/20/2010 | 668 | 60 / 61 | 569 | 0.9/1.4/1.8 | 0/21/756 |
| 47 | 121_92 | 92 | 10/20/2010 | 668 | 60 / 61 | 591 | 1.3/2.0/2.5 | 0/30/1058 |
| 48 | 121_185 | 185 | 10/20/2010 | 668 | 60 / 61 | 412 | 1.8/2.8/3.4 | 0/60/1400 |
| 49 | 121_370 | 370 | 10/20/2010 | 668 | 60 / 61 | 471 | 2.5/3.9/4.5 | 0/75/1571 |
| 50 | 121_740 | 740 | 10/20/2010 | 668 | 60 / 61 | 476 | 3.5/5.1/6.1 | 0/109/1566 |
| 51 | 130_46 | 46 | 10/30/2000 | 668 | 60 / 61 | 519 | 0.9/1.4/1.8 | 0/21/286 |
| 52 | 130_92 | 92 | 10/30/2000 | 668 | 60 / 61 | 457 | 1.3/2.0/2.4 | 0/40/968 |
| 53 | 130_185 | 185 | 10/30/2000 | 668 | 60 / 61 | 476 | 1.8/2.8/3.4 | 0/61/2281 |
| 54 | 130_370 | 370 | 10/30/2000 | 668 | 60 / 61 | 506 | 2.6/3.8/4.5 | 0/75/1574 |
| 55 | 130_740 | 740 | 10/30/2000 | 668 | 60 / 61 | 464 | 3.5/5.1/5.9 | 0/115/1789 |
| 56 | 202_46 | 46 | 20 / 0 / 20 | 668 | 60 / 61 | 670 | 0.9/1.4/1.8 | 0/20/656 |
| 57 | 202_92 | 92 | 20 / 0 / 20 | 668 | 60 / 61 | 545 | 1.3/2.1/2.4 | 0/36/1550 |
| 58 | 202_185 | 185 | 20 / 0 / 20 | 668 | 60 / 61 | 527 | 1.8/2.8/3.3 | 0/50/1790 |
| 59 | 202_370 | 370 | 20 / 0 / 20 | 668 | 60 / 61 | 543 | 2.5/3.8/4.4 | 0/73/1772 |
| 60 | 202_740 | 740 | 20 / 0 / 20 | 668 | 60 / 61 | 501 | 3.5/5.2/5.9 | 0/111/1800 |
| 61 | 211_46 | 46 | 20 / 10 / 10 | 668 | 60 / 61 | 501 | 0.9/1.5/1.8 | 0/25/680 |
| 62 | 211_92 | 92 | 20 / 10 / 10 | 668 | 60 / 61 | 395 | 1.3/2.0/2.5 | 0/44/896 |
| 63 | 211_185 | 185 | 20 / 10 / 10 | 668 | 60 / 61 | 440 | 1.8/2.8/3.4 | 0/65/1571 |
| 64 | 211_370 | 370 | 20 / 10 / 10 | 668 | 60 / 61 | 504 | 2.5/3.8/4.5 | 0/82/1770 |
| 65 | 211_740 | 740 | 20 / 10 / 10 | 668 | 60 / 61 | 469 | 3.5/5.1/6.1 | 0/121/1965 |
| 66 | 220_46 | 46 | 20 / 20 / 0 | 668 | 60 / 61 | 579 | 0.9/1.4/1.8 | 0/25/602 |
| 67 | 220_92 | 92 | 20 / 20 / 0 | 668 | 60 / 61 | 574 | 1.3/2.0/2.5 | 0/39/1541 |
| 68 | 220_185 | 185 | 20 / 20 / 0 | 668 | 60 / 61 | 551 | 1.8/2.7/3.2 | 0/59/968 |
| 69 | 220_370 | 370 | 20 / 20 / 0 | 668 | 60 / 61 | 515 | 2.6/3.8/4.4 | 0/73/1348 |
| 70 | 220_740 | 740 | 20 / 20 / 0 | 668 | 60 / 61 | 446 | 3.5/5.2/5.8 | 0/114/1860 |

*Each run name is of the form: nnn_p. The first 3 'n's refer to the coverages in 10ths of the 3 different thicknesses.
The p number is the thrust level in KN. All runs were a transit of a 500 m long ice field ( x 200 m wide).
** min, mean and max values

The ice floes are represented as convex polygons with a range of apex angles. The angles for all 668 floes were analyzed to examine the distribution of the values. As shown in Figure 6.31, the angles appear to follow a Weibull distribution, though not

perfectly. One interesting aspect is that the angles are limited to 180 degrees. The Weibull distribution appears to fit the data quite well, but fails to capture the fixed upper limit at 180. As can be seen from Figure 10, the Weibull model would predict that a small number of apex values would be above 180 degrees.

While this is obviously impossible (for convex shapes), the model appears to fit the bulk of the data quite well. This statistical modeling was performed using the Minitab software (Minitab 2013). The reason for presenting these values is that the floe apex angle is one of the key input parameters that determines the impact force values. The higher apex angles result in higher force values. This relationship may be counter-intuitive. The reason is that higher angles mean a more rapid rise in area and force as contact occurs, resulting in a 'harder' impact.

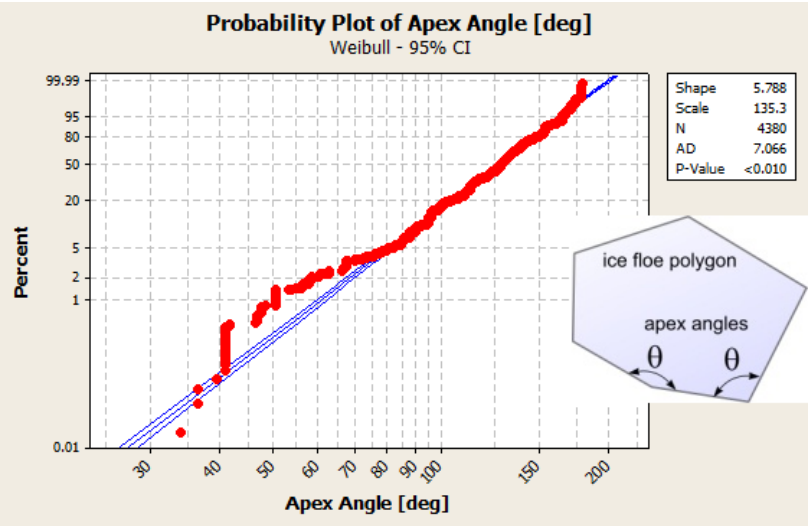



Figure 6.31: Probability plot for ice floe apex angle data.[10]

Another important input parameter is the ice floe mass. Figure 6.32 shows the mass statistics for all 668 floes and also for the set of 2520 impacted floes that occurred

in runs 46-50. The floe mass is determined by the product of area, thickness and mass density. The mass values appear to follow a lognormal distribution. It appears that the floes impacted are representative of the whole population. This would be expected in the case of the simple navigation strategy modeled here. If a more sophisticated hazard avoidance strategy were to be modeled one might expect a different result. The distributions of apex angle and floe mass are the result of the shapes and sizes of the ice floes in the digitized image (Figure 6.8), rather than being user determined.

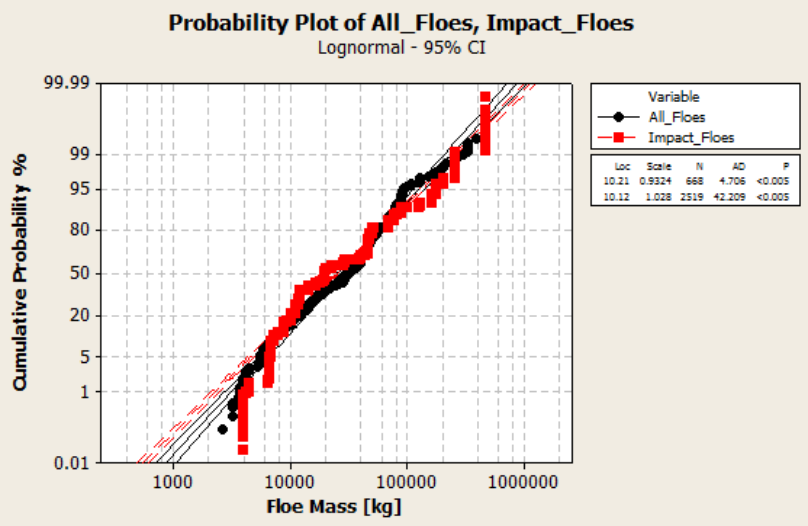



Figure 6.32: Probability plot for ice floe mass values, for both all floes and just those floes struck in runs 46-50.[10]

#### 6.2.2.3 Parametric Results

There are various kinds of parametric simulation results that will be presented below. These particular results are from runs 46-50, which involve 10% thick ice, 20% medium ice and 10% thin ice. The five runs are for a range of power levels and velocities, and cover 2.5km of transit.

Figure 6.33 shows the set of locations of the impacts on the bow. The points tend to be on the hull edge, though there are cases where contact could appear to be inside or outside the hull. This is because of the way contact is defined, as is sketched in the figure. There tend to be a greater number of impacts towards the stem. Figure 6.34 quantifies this trend by plotting the percentage of impacts that occur within each meter of width of the vessel. In a simple estimate of the rate of impacts per meter width, one might expect that the rate per meter would be constant. This is because each meter will sweep through the same area of ice cover and nominally sweep over the same number of floes (assuming a uniform ice cover as in this case). However, the actual kinematics of the collisions tend to result in the more forward collisions creating a shadow or shield that lowers the number of collisions further aft. This trend might change significantly if more complex navigation practices were to be modeled. The navigation here was just a simple auto pilot with no attempt to avoid any specific features.

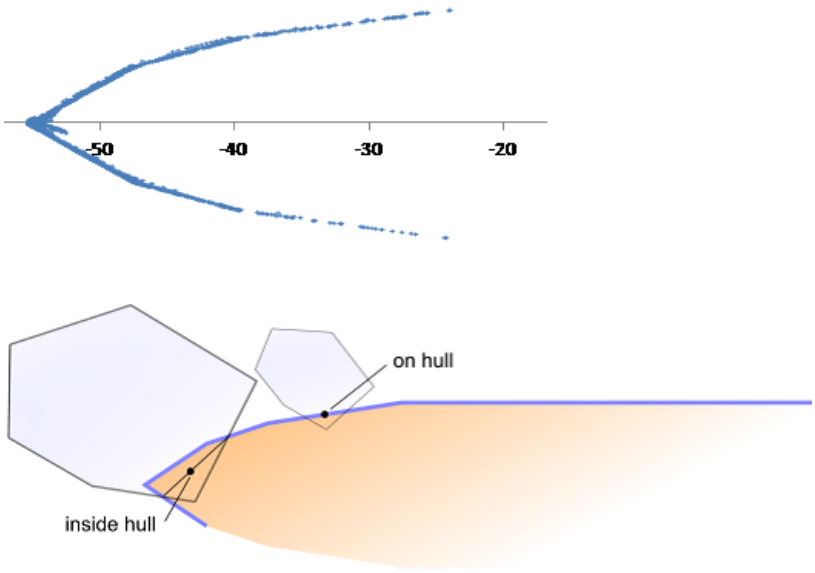



Figure 6.33: Plot of impact locations on the vessel (runs 46-50).[10]

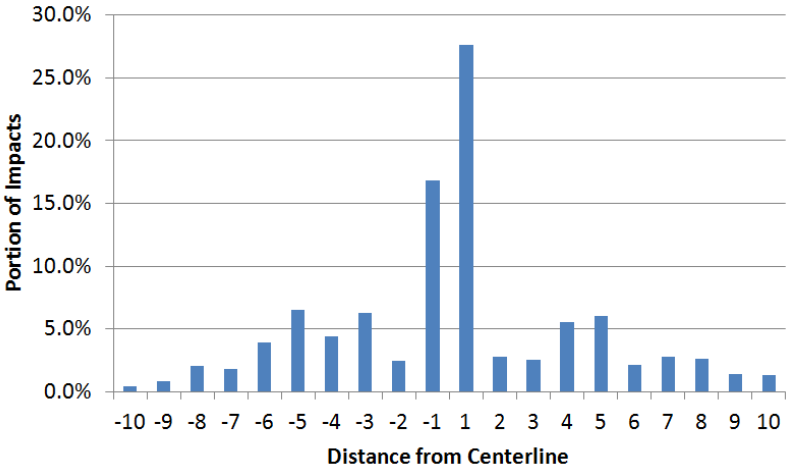



Figure 6.34: Plot of % impacts vs. lateral distance from centerline (runs 46-50). [10]

Figure 6.35 plots the magnitude of the impact forces vs. the distance from the stem. This shows the maximum forces occur closer to the stem. The specific shape of the vessel (waterline and frame angles) will influence these results, possibly strongly.

In this work only one hull form has been examined.

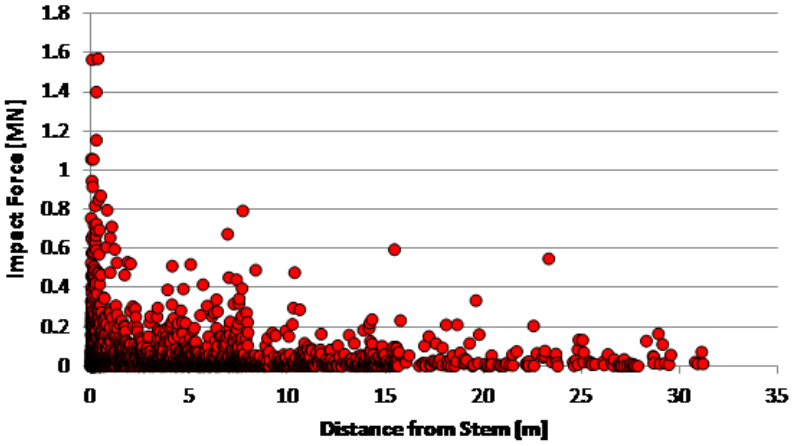



Figure 6.35: Plot of impacts forces vs. distance from stem (runs 46-50). [10]

Figure 6.36 plots the magnitude of the impact forces vs. the vessel speed. The data has the typical appearance of field data (see Figure 6.41) where trends are easily obscured by the mass of variable data. The variations are also influenced by hull shape, floe size, thickness and apex angle. The effect of velocity can be obscured. A trend line through the data is most strongly influenced by the majority of small impacts. The equation relating mean force to velocity is:

$$F = 0.0023v^{1.68} \text{ [MN]}$$

The higher values of force appear to be following a somewhat different trend, in that they appear to be limited to a force of 1.6 MN. This is obviously an artifact of the specific simulation rather than an actual limit. The load mechanics used in the simulation are deterministic and as such the forces should be bounded. In most impacts the various input parameters combine to produce load lower than the

maximum.

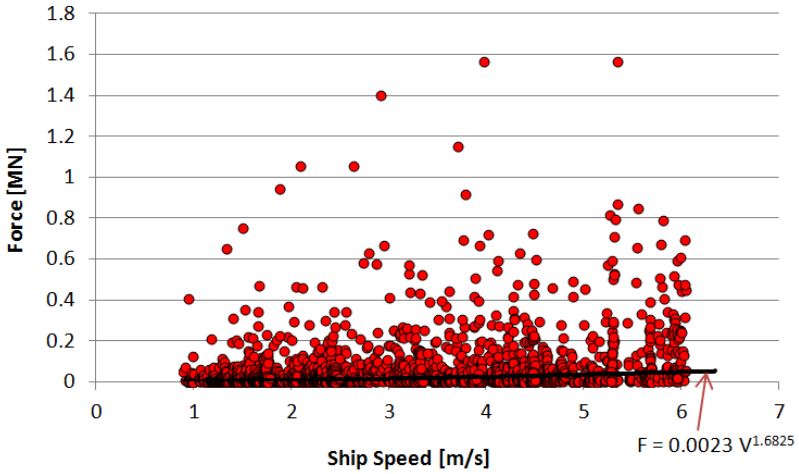



Figure 6.36: Plot of impacts forces vs. ship speed (runs 46-50).[10]

Figure 6.37 presents results for a specific subset of the collisions. Only those impacts on the first panel of the hull near the bow, and only those involving an impact with 0.7m thick floes are presented. Along with the GEM data is the solution for force vs. velocity for a collision with a 252t floe, with a 170deg. apex angle, both of which approximately represent the highest possible values in the simulation. This is a worst case combination, for which the flexural failure limit is also included. The data lies well within the bounds of the limit case.

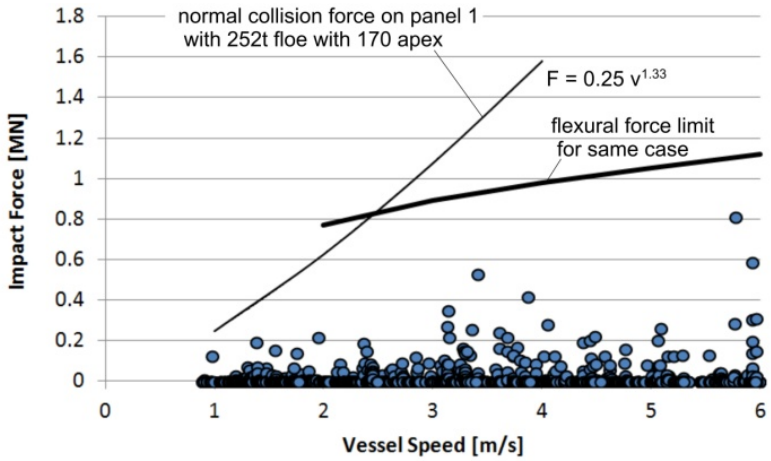



Figure 6.37: Plot of impacts forces vs. ship speed on panel 1 for 0.7m thick floes (runs 46-50).[10]

#### 6.2.2.4 Load Level Statistics

The ice load statistics for several groups of runs appear to follow a Weibull distribution, especially at the upper end. Figure 6.38 shows the cumulative probability distribution data for a set of cases. Data labelled 121 is from runs 46-50. Data labelled 004 is from runs 16-20. Data labelled 040 is from runs 21-25. Data labelled 400 is from runs 26-30. The set labelled All is all of the above. In each case the coverage is 40%. In the case of 004 the ice thickness is 0.5m, or thin ice. In the 040 case the ice is all 0.7m thick, or medium ice. In the 400 case the ice is all 1.2m thick, or thick ice. In the 121 case there is 10% thick ice, 20% medium ice and 10% thin ice. In all cases the data has been modeled with a Weibull distribution, which has a cumulative distribution function:

$$F(x, k, \lambda) = 1 - e^{x/\lambda}$$

where $x$ is the load in Newtons, $k$ is the shape parameter and $\lambda$ is the scale parameter. Figure 6.38 shows that the distributions are all very similar, though not identical as can be seen by examining the scale parameter for each data set. Loads are higher in the thicker ice, as would be expected. The remarkable thing is that the overall variation of the loads tend to mask the relatively small variations caused by thickness changes. The other sources of variation include velocity, floe size, floe apex angles and hull angles

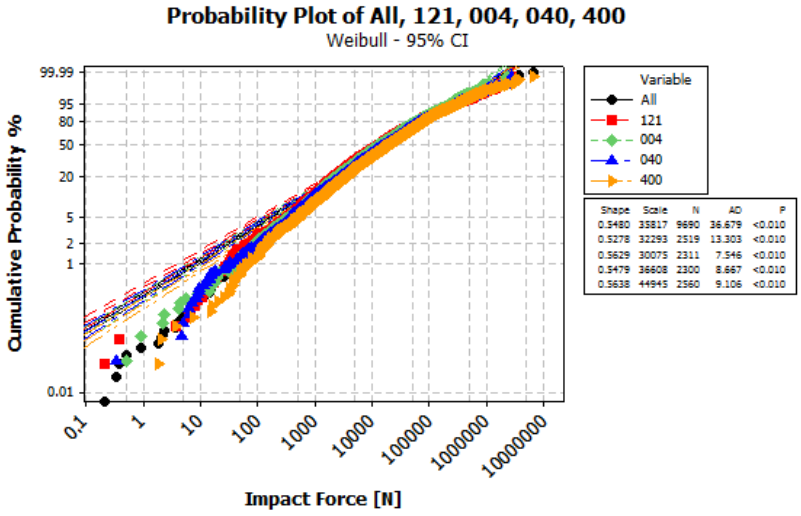



Figure 6.38: Probability plots of cumulative distribution of impacts forces (runs 16-30 and 46-50).[10]

#### 6.2.2.5 Ice Load Statistics from Fields Trials Data

Ice impact load data has been gathered on a number of vessels. Figure 6.39 shows a map of the areas where ice impact load data have been collected on four different vessels. The USCGC Polar Sea conducted a series of western arctic trials in the 1980s [15, 58, 13, 14]. The Polar Sea had a load panel installed in its bow, large enough to

capture impact loads as the ship struck ice floes. The data from those trials covers a wide variety of ice conditions, ranging from first year ice in the Bering Sea to heavy multi-year ice in the North Chukchi and Beaufort Seas, covering everything from open pack, to close pack, ridged and level ice. The CCGS Louis S. St. Laurent conducted a trans-Arctic voyage in 1997 and measured impact loads on panels similar to the arrangement on the Polar Sea [54]. The Polar Sea and Louis St. Laurent data had similar load measuring systems, and could measure the total impact force during a collision with an ice edge.

A Baltic ice class vessel called the MS Kemira was instrumented to measure frame loads [34, 35]. The Kemira data was collected during normal commercial cargo voyages in first year sea ice in the northern Baltic. The KV Svalbard, a Norwegian Coast Guard vessel, was also instrumented to measure frame loads. Data was recorded in the Barents Sea in 2007 [37, 36].

The data collected on these vessels represents a significant portion of the available scientific data concerning ice impact loads in sea ice. The data from these various trials will be discussed. It is important to consider that the vessels were of differing size and shape, operating in differing conditions, and with quite different sensor packages and levels of coverage


Figure 6.39: Arctic region map showing various ice loads ship trials.[10]

The present authors have direct access to the raw measurement data from both the Polar Sea and the Louis S. St. Laurent. The authors do not have access to the raw data from the Kemira or Svalbard. However, analysis of the data from the Kemira and Svalbard has been published. The authors of those data sets have suggested that the data follows Weibull or similar (i.e. exponential) distributions. See [60] for analysis of Kemira data and [1] for a discussion of Svalbard data.

Figure 6.40 shows some of the impact load data from the Polar Sea plotted as impact force vs. ship speed. Within and one sea area there appears to be little obvious relationship between force and velocity, much as was observed in the GEM simulation (see e.g. Figure 6.36). It is interesting to note that in sea areas with lighter

ice (Bering Sea) the vessel speeds were higher while the loads were lower than were the case in the regions of heavier ice (North Chukchi Sea). This is a natural result. For the present assessment it shows that loads are influenced by a combination of ice conditions and navigation practices.

Figure 6.41 shows impact load data from the 1994 Arctic Ocean voyage of the Louis S. St. Laurent. Once again there is no obvious trend between force and velocity, with a very slight inverse relationship when a single curve is fit to all data. The vessel transited a wide variety of conditions and so would have experienced similar navigation effects as discussed above.

It should be noted that the field data from the two vessels is subject to a number of artifacts that GEM data is not. Field data tends to be gathered with a threshold, such that all small load values are ignored. Also there is the problem of the completeness of the record. For both the Polar Sea and the St. Laurent, some of the load data did not have a corresponding velocity. All such data was plotted at a small velocity (0.25m/s), which does obviously involve an error. The GEM simulation values are complete in all respects, with all impacts at all locations fully logged.

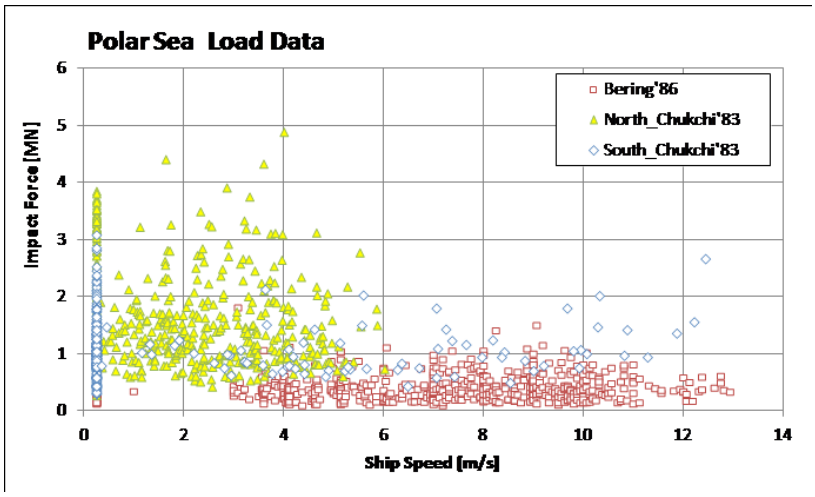



Figure 6.40: Ice impact load vs. ship speed from USCG POLAR SEA during 3 voyages.[10]

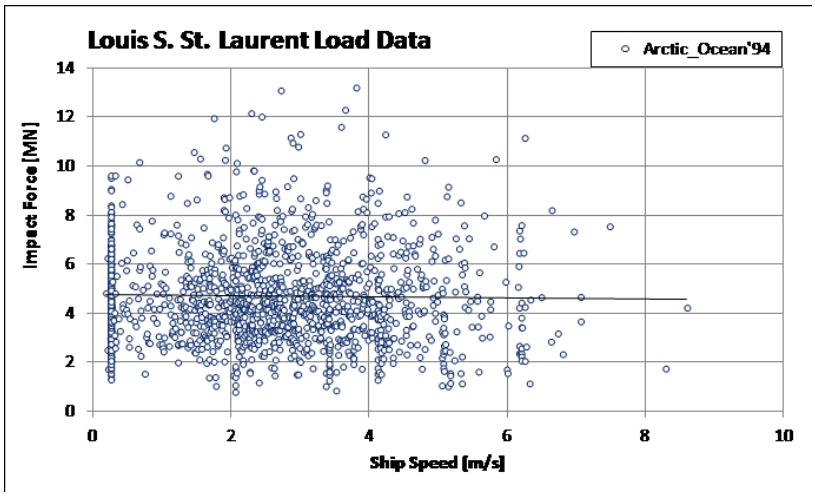



Figure 6.41: Ice impact load vs. ship speed from CCGS LOUIS S ST. LAURENT during a trans arctic voyage in 1994.[10]

Figure 6.42 shows one probability distribution for ice impacts on the Polar Sea in first year ice in the South Bering Sea. The data appears to show fluctuations which may be associated with varying ice conditions and interaction mechanisms.

Nevertheless, the data is reasonably well described by a Weibull distribution.

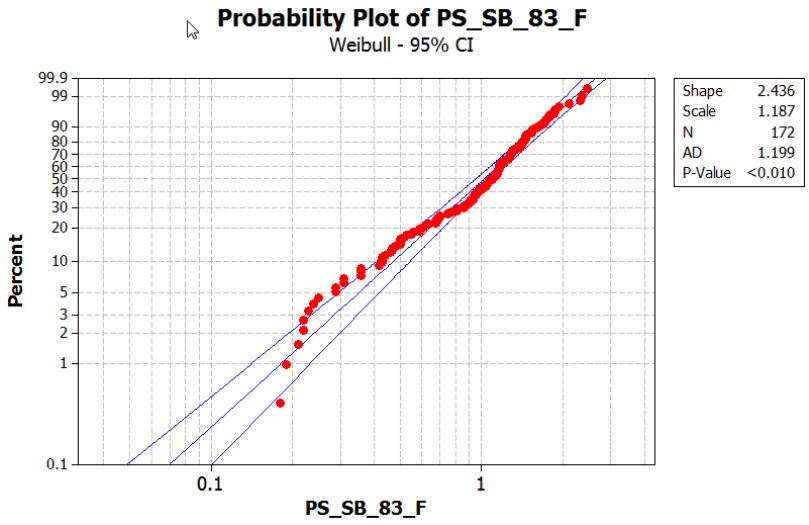



Figure 6.42: Ice impact load statistics for the POLAR SEA during its 1983 voyage in the first year ice of the south bering sea.[10]

#### 6.2.2.6 Discussion

Ship-ice interaction is a complex process, influenced by many nonlinear and some linear mechanical processes as well as by the many vessel design parameters and the navigation practices. Developing an understanding of the process is a challenge that requires the integration of many approaches. Full scale data is crucially needed to provide direct knowledge of the process and to allow validation of the models and theories used to describe the process. Unfortunately full scale data is both limited and imperfect. Conventional numerical modeling approaches have tended to focus on either the local mechanics or the broad system level, often leaving these two types of models somewhat disconnected.

In this work we have presented results from a new development we call GEM

(GPU-Event Mechanics). The approach allows us to follow the movement of the vessel (s) and the ice floes for a long period of time, even while we include all the individual collision and contact events. By combining the modeling of short term events and long term kinematics, the model accounts for system level behaviour without the need to overly simplify the kinematics and impacts. It is intended to expand on the range of events covered and to improve the sophistication of the kinematics.

This work presents a number of new insights into some questions of interest. One is the question of the statistical nature of ice loads. This analysis has shown that while ice thickness does influence load, through its influence of mass and flexural strength, the main cause of variations shown here is due to the variable ice mass and apex angle. While this is far from definitive, it is a useful insight. In many situations the ice thickness does not vary over orders of magnitude while the loads often do. The GEM program can be a useful tool in exploring the sources of variability in the loads, helping to establish a better understanding of the statistics, especially at the extreme or design levels.

Another useful result is shown in Figure 6.34. While one might expect that a ship in uniform pack ice would experience a similar impact rate per meter of breadth anywhere in the bow, the GEM results are showing a kind of shadowing effect. This is possible because all ice floe motions and interactions are being tracked. Further studies of a wider range of ice conditions, combined with more realistic navigation strategies would help to explain both the rate of collisions and also the appropriate design loads for various parts of a vessel. The question of the validity of the hull area

factors used in ice class structural design is of great practical significance.

Figures 6.36,6.37 show the GPU results for force as a function of speed. As is typical of full scale data (Figures 6.40,6.41) the effects of speed are lost in the general scatter. The field data and even the GEM data show no obvious limits (upper bounds). Nevertheless, the GEM model mechanics have very specific limits that are so rarely reached that they are not evident. Most statistical models assume open tail distributions, and so may predict extreme design values higher than may be physically possible. The GEM model can easily be used with probabilistic as well as deterministic inputs, and would be able to explore this question, and remove unnecessary conservatism. Any excessive conservatism, is costly and tends to undermine potential improvements in other aspects of a design

# Chapter 7

# Conclusion and Future Work

## 7.1 Conclusion

The experiments demonstrated performance benefits for simulating the complex mechanics of a ship operating in pack ice. It is clear that GPGPU has the potential to significantly improve the processing time of highly data parallel algorithms.

The discussion and results have described a new class of model that integrates a number of old ideas into a new capability. The developments of the ice simulator in this thesis have permitted the modeling of a massive event set in faster than real time, using affordable desktop computer hardware. With demands for greater safety and greater understanding of ship and structure operations in polar regions, there is a need for new simulation tools. The Ice-Event-Mechanics approach permits the user to model complex problems in a timely and practical way.

The numerical model shows the general trends which are also visible in the exper-

imental data. Especially in the pack ice scenario, it shows realistic behaviour.

### 7.1.1 Contributions

The main contributions of this work are as follows:

- It introduces a new GPGPU simulator that can be used to simulate the behaviour of a ship in pack ice. Using the new GPGPU simulator, hyper-real-time simulations can be achieved which will allow the results to be used in planning ice management activities. This feature has great practical significance for design, assessment and training applications.

- It demonstrates three different general structures of the GPGPU solution for the ice simulator.

- It describes a GPGPU implementation of a collision detection algorithm (Separating Axes Theorem) which can be used to detect the collisions between convex polygons.

- It describes a GPGPU implementation of an algorithm that can be used to find the intersection between convex polygons.

- It describes a GPGPU implementation of an algorithm that can be used to triangulate a simple polygon (Polygon Triangulation By Ear Clipping).

- It demonstrates the performance of using different GPGPU collision detection approaches (Triangulation Mesh and Uniform Grid Structure) to detect the

collisions between polygons.

- It introduces a new GPGPU collision detection approach (variable radius) to reduce the number of polygons to be checked for collision by eliminating those that are beyond some distance away.

## 7.2 Future Work

Further development and optimization are necessary for a larger ice fields. One way to improve the performance of the simulation for a larger ice fields is to implement the simulator using multiple GPUs.

The GEM model is a work in progress. The version discussed here tracks a single vessel through a simple open ice pack and it has the following features:

- Floe edge flexural failure, with new floe creation.
- Wind loads on floes.
- Current forces on floes.

Further enhancements are being planned that will add:

- Rafting behaviour (2.5D).
- Floe Splitting.
- Simplified Ridging at floe-floe contacts.

The above enhancements can be implemented in the current 2.5D model. To take the technology to an entirely new level, the modeling will need to be implemented in a full 3D framework.

In 2012, Intel announced that Xeon Phi will be the brand name used for all products based on their Many Integrated Core architecture. An interesting experiment to be done in the future is implement the simulator using the Xeon Phi and compare the performance with the Tesla GPU card.

With an improved model, an improved method also needs to be found to validate the model through model experiments. This should include better controlled ship motions (so that sway and yaw motions are resisted), a more realistic representation of ice floes and a more effective quantitative method to compare the trajectories between the experiments and GPGPU simulations.

# Appendix A

# Appendix

## A.1 GPU Ice Simulation Equations

This section describes the GPU ice simulation equations that have been implemented in the ice simulator. These equations derived by Dr. Claude Daley. The units used in this work follow the international system of units (SI).

### A.1.1 Polygon Geometry

Each polygon is defined by the coordinates of **n** points. These points are in counter-clockwise. Each point can be expressed in one of 3 coordinate systems, global, local(aligned) and local(rotated). All three are used at various points.

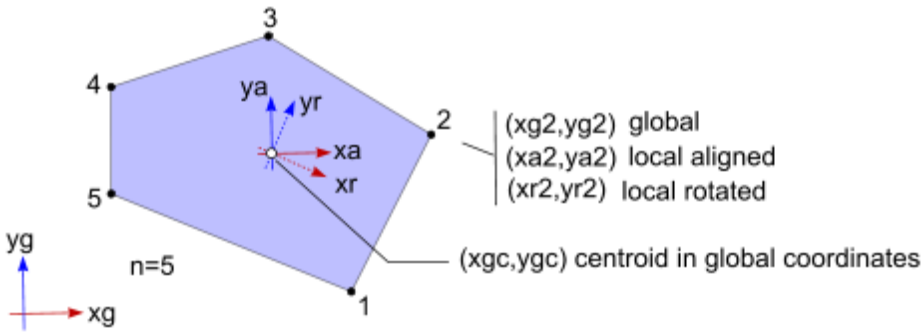



Figure A.1: Polygon coordinate systems.

#### A.1.1.1 Area

The area of any polygon **A** will not change until the floe is broken into two parts. So the area can be calculated from the global coordinates and stored. The calculation will work using any of the 3 coordinate systems.

$$ag_i = (xg_i yg_{i+1} - xg_{i+1} yg_i) \tag{A.1}$$

$$A = \frac{1}{2} \sum_{i=1}^{n} ag_i \tag{A.2}$$

(note: for i=n+1, use i=1, $ag_i$ is the signed swept area of the line defined by each pair of points).

#### A.1.1.2 Centroid Coordinates

$$xg_c = \frac{1}{6A} \sum_{i=1}^{n} (xg_i + xg_{i+1}) ag_i \tag{A.3}$$

$$yg_c = \frac{1}{6A} \sum_{i=1}^{n} (yg_i + yg_{i+1}) ag_i \tag{A.4}$$

The calculation will only work using the global coordinate system. It will produce zero for local coordinates (may be used as a check).

#### A.1.1.3 Local Polygon Coordinates (Aligned)

Local (aligned) coordinates are found by shifting the values by the global centroid coordinates:

$$xa_i = xg_i - xg_c \tag{A.5}$$

$$ya_i = yg_i - yg_c \tag{A.6}$$

#### A.1.1.4 Polygon Aligned Area Moments

A general polygon has two 2nd moments of inertia as well as a product of inertia:

$$J_{xx} = \frac{1}{12}\sum_{i=1}^{n}(ya_i^2 + ya_i ya_{i+1} + ya_{i+1}^2)aa_i \tag{A.7}$$

$$J_{yy} = \frac{1}{12}\sum_{i=1}^{n}(xa_i^2 + xa_i xa_{i+1} + xa_{i+1}^2)aa_i \tag{A.8}$$

$$J_{xy} = \frac{1}{24}\sum_{i=1}^{n}(xa_i ya_{i+1} + 2xa_i ya_i + 2xa_{i+1}ya_{i+1} + xa_{i+1}ya_i)aa_i \tag{A.9}$$

Where $aa_i = (xa_i ya_{i+1} - xa_{i+1}ya_i)$.

These values must be computed using the aligned local coordinates.

#### A.1.1.5 Polygon Principal Axis Angle

The angle $\theta$ is the angle from the global x axis to the principal axis:

$$\theta = -\frac{1}{2} atan(\frac{2J_{xy}}{J_{xx} - J_{yy}}) \tag{A.10}$$

#### A.1.1.6 Local Polygon Coordinates (Rotated)

$$xr_i = xa_i \cos\theta + ya_i \sin\theta \tag{A.11}$$

$$yr_i = -xa_i \sin\theta + ya_i \cos\theta \tag{A.12}$$

#### A.1.1.7 Polygon Principal Area Moments

A general polygon has two 2nd moments of inertia as well as a product of inertia:

$$I_{xx} = \frac{1}{12} \sum_{i=1}^{n} (yr_i^2 + yr_i yr_{i+1} + yr_{i+1}^2) ar_i \tag{A.13}$$

$$I_{yy} = \frac{1}{12} \sum_{i=1}^{n} (xr_i^2 + xr_i xr_{i+1} + xr_{i+1}^2) ar_i \tag{A.14}$$

$$I_{xy} = \frac{1}{24} \sum_{i=1}^{n} (xr_i yr_{i+1} + 2xr_i yr_i + 2xr_{i+1} yr_{i+1} + xr_{i+1} yr_i) ar_i \tag{A.15}$$

$$I_{zz} = I_{xx} + I_{yy} \tag{A.16}$$

Where $ar_i = (xr_i yr_{i+1} - xr_{i+1} yr_i)$. These values must be computed using the rotated (principal) local coordinates.

### A.1.2 The Vessel Properties

The mass of the ship can be found as:

$$M = L.B.T.C_B.\rho_{water} \tag{A.17}$$

Where $L$ is the legnth of the ship = 100 m, $B$ is the beam of the ship = 20 m, $T$ is the height of the ship = $\frac{B}{2.5}$ = 8 m, $C_B$ is the block coefficient = 0.7 and $\rho_{water}$ = 1025 kg/m$^3$.

The principal area moments of the ship can be found as:

$$I_x = Mrx^2 \tag{A.18}$$

$$I_y = Mry^2 \tag{A.19}$$

$$I_z = Mrz^2 \tag{A.20}$$

Where $rx = 0.33B$, $ry = 0.25L$, $rz = 0.25L$ and $M$ is the mass of the ship.

### A.1.3 Description of the Ice Floes Motion

Polygon movement is represented by movement of the centroid $g$. At $t_j$: $g_j = (X_{gj}, Y_{gj}, r_j)$. Where $(X_{gj}, Y_{gj})$ are the global X and Y coordinates of the centroid and $r_j$ is the angular position of the centroid.

For each polygon, there are three positions $X_{gj}, Y_{gj}$ and $r_j$. Also, there are three velocities $V_x, V_y$ and $\omega$. In this case, after a time change $\Delta t$:

$$X_{gj+1} = X_{gj} + V_x \Delta t \tag{A.21}$$

$$Y_{gj+1} = Y_{gj} + V_y \Delta t \tag{A.22}$$

$$r_{gj+1} = r_{gj} + \omega \Delta t \tag{A.23}$$

Given $g_j$, the local coordinates of each polygon can be as:

$$XL_i = X_i - X_{gj} \tag{A.24}$$

$$YL_i = Y_i - Y_{gj} \tag{A.25}$$

for all i (all the points). Then, the orthogonal coordinates can be found as:

$$R_i = \sqrt{XL_i^2 + YL_i^2} \tag{A.26}$$

$$\theta_i = atan2(YL_i, XL_i) \tag{A.27}$$

Then, the angle $\theta$ can be updated as:

$$\theta_i = \theta_i + (r_{j+1} - r_j) \tag{A.28}$$

Finally, the global coordinates can updated as:

$$X_i = (R_i \cos(\theta_i)) + X_{gj} \tag{A.29}$$

$$Y_i = (R_i \sin(\theta_i)) + Y_{gj} \tag{A.30}$$

### A.1.4 Description of the Mass Reduction Coefficient Co for the 2D Simulation

A collision taking place at point 'P' (see Figure A.2) on the ship, will result in a normal force Fn. Point P will accelerate, and a component of the acceleration will be along the normal vector **N**, with a magnitude $a_n$. The collision can be modeled as if point P were a single mass (a 1 degree of freedom system) with an equivalent mass Me. The equivalent mass is a function of the inertial properties (mass, radius of gyration, hull angles and moment arms) of the ship. The equivalent mass is linearly proportional to the mass (displacement) of the vessel, and can be expressed as:

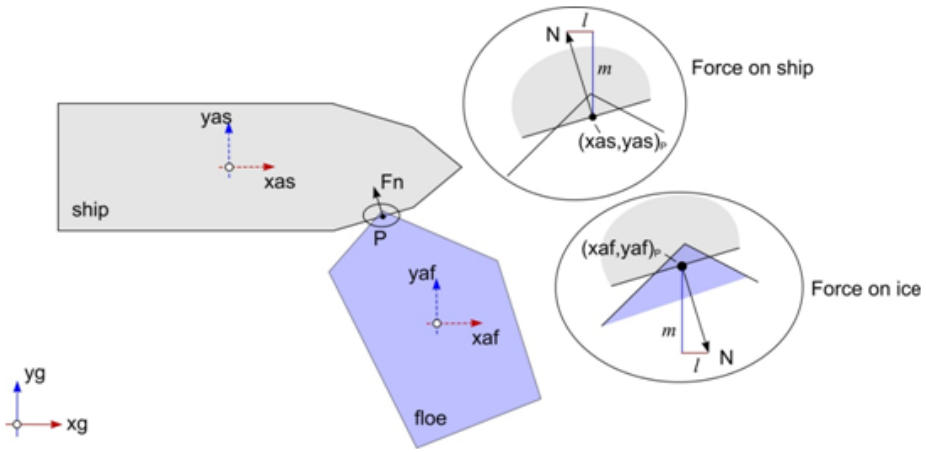



Figure A.2: 2D ship collision point geometry.

$$Me = M/Co \tag{A.31}$$

Where Co is the mass reduction coefficient. This approach was first developed by Popov (1972).

At any point P ($x_{as}$,$y_{as}$) on a 'wall sided' hull ($\beta = 0$) the inward normal vector can be expressed as:

$$N = li + mj \tag{A.32}$$

Where $i$ and $j$ are the unit vectors aligned with the global coordinate system, $l$ and $m$ are called 'direction cosines' and can be positive or negative.

The yaw moment arm is:

$$\eta l = mxs - lys \tag{A.33}$$

In this case, with added masses ignored, the mass radius of gyration (squared) in yaw is:

$$rz^2 = Iz/A \tag{A.34}$$

With the above defined, the mass reduction coefficient is:

$$Co = l^2 + m^2 + \eta l^2/rz^2 \tag{A.35}$$

### A.1.5 The Impulse Calculation

The collision applies an impulse $I_e(= Fndt)$ to the floe at the point of contact. There are two types of impluse: Elastic and InElastic.

The InElastic impluse is:

$$I_{InElastic} = M_{ec}V_c \tag{A.36}$$

$$M_{ec} = \frac{1}{\frac{1}{M_{ec1}} + \frac{1}{M_{ec2}}} \tag{A.37}$$

$$M_{ec1} = M_1/Co_1 \tag{A.38}$$

$$M_{ec2} = M_2/Co_2 \tag{A.39}$$

$$V_c = V_{n1c} + V_{n2c} \tag{A.40}$$

$$V_{nc1} = -V_{xg1}l_{1c} - V_{yg1}m_{1c} - \omega_{g1}\eta_1 \tag{A.41}$$

$$V_{nc2} = -V_{xg2}l_{2c} - V_{yg2}m_{2c} - \omega_{g2}\eta_2 \tag{A.42}$$

Where $M_{ec}$ is the effective mass of the collision, $M_{ec1}$ is the effective mass of floe 1, $M_{ec2}$ is the effective mass of floe 2, $M_1$ is the mass of floe 1, $M_2$ is the mass of floe 1, $Co_1$ is the mass reduction coefficient of floe 1, $Co_2$ is the mass reduction coefficient of floe 2, $V_c$ is the closing velocity along the normal N, $V_{n1c}$ is the outward velocity

along N for floe 1 at the centroid, $V_{n2c}$ is the outward velocity along N for floe 2 at the centroid, $V_{xg1}, V_{yg2}$ is the linear velocity for floe 1 at the centroid, $V_{xg2}, V_{yg2}$ is the linear velocity for floe 2 at the centroid, $\omega_{g1}, \omega_{g2}$ is the angular velocity for floe 1,2 at the centroid and $\eta_1, \eta_2$ is the yaw moment arm for floe 1,2.

The Elastic impulse is:

$$I_{Elastic} = 0.05C\sigma_{max}wh\Delta t \tag{A.43}$$

$$C = min(((A_0K_{ice})/w * D), 1) \tag{A.44}$$

$$h = min(h_1, h_2) \tag{A.45}$$

$$D = min(sqrt(A_1), sqrt(A_2)) \tag{A.46}$$

Where $w$ is the length of the line of intersection, $\Delta t$ is the time step, $h1, h2$ are the thicknesses of the two floes, $A_1, A_2$ are the areas of the two floes, $A_0$ is the area of the overlap between the two floes, $\sigma_{max} = 1$Mpa and $K_{ice} = 100$ units.

When the friction is ignored, all contact forces are normal to lines of contact. When the friction is included, a second force is added in the tangential direction.

The changes in velocity at the center of gravity in the globally aligned local coordinate system are:

$$dV_x = (I_e l)/M \tag{A.47}$$

$$dV_y = (I_e m)/M \tag{A.48}$$

$$dV_\omega = (I_e \eta l)/(Mrz^2) \tag{A.49}$$

### A.1.6 Description of the Mass Reduction Coefficient Co for a Ship in 2.5D Simulation

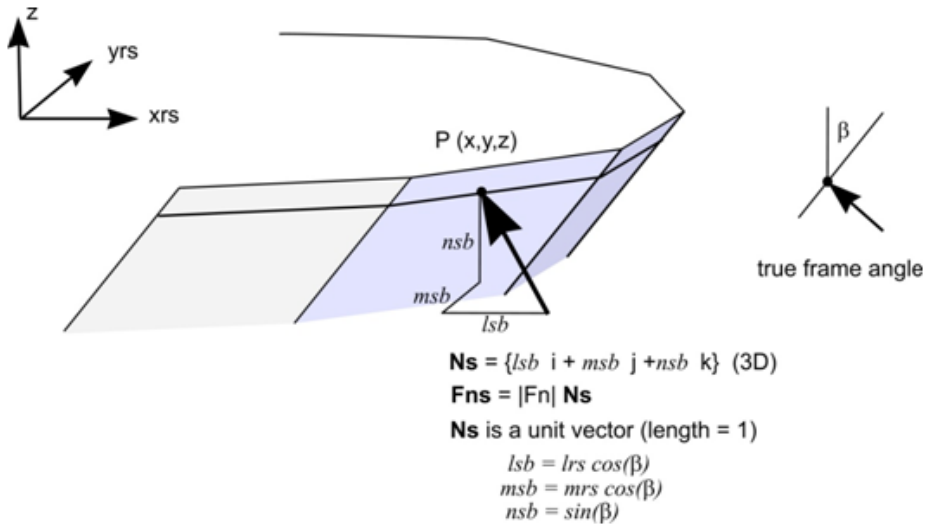



Figure A.3: 2.5D ship collision point geometry.

The coordinate system shown in Figure A.2 is valid of 2D (wall-sided) collisions. For cases where the ship is considered to have sloping sides as shown in Figure A.3, there are two improvements that are required. On improvement is the use of rotated (principal) coordinates as shown in Figure A.4. One reason for using principal coordinates is that the roll and pitch responses of the ice floe are better handled using the principal coordinates. Secondly, it will be easier to calculate and express the 3D normal vector if we use the principal exes.

At any point P on a 'sloped' hull ($\beta > 0$) the inward Normal 2D vector Ns can be expressed as:

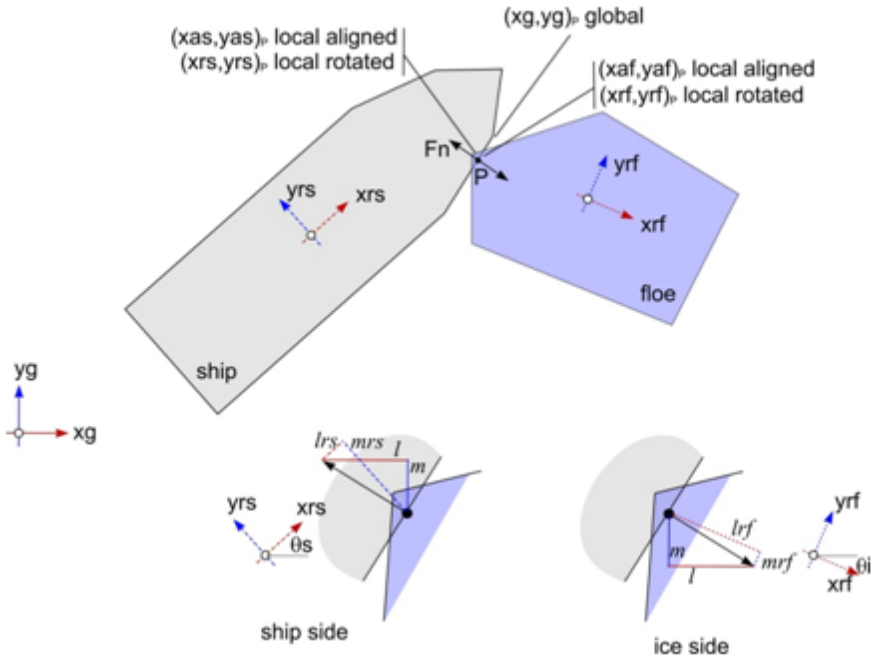



Figure A.4: Rotated x-y coordinate systems for ship and ice.

$$\mathbf{Ns} = li + mj \tag{A.50}$$

The same vector can be re-expressed in rotated coordinates:

$$\mathbf{Nsr} = lrsi + mrsj \tag{A.51}$$

$$lrs = l\cos(\theta) + msin(\theta) \tag{A.52}$$

$$mrs = -l\sin(\theta) + mcos(\theta) \tag{A.53}$$

The 3D normal vector **Nsb** can be derived from **Nsr**:

$$\mathbf{Nsb} = lsbi + msbj + nsbk \tag{A.54}$$

$$lsb = lrs\cos(\beta) \tag{A.55}$$

$$msb = mrs\cos(\beta) \tag{A.56}$$

$$nsb = sin(\beta) \tag{A.57}$$

And the moment arms are (assume z=0):

$$\lambda ls = nsbyrs - msbz \tag{A.58}$$

$$\mu ls = lsbz - nsbxrs \tag{A.59}$$

$$\eta ls = msbxrs - sbyrs \tag{A.60}$$

The added mass terms are as follows (estimates):

$$AMsx = 0 \tag{A.61}$$

$$AMsy = 0.5 \tag{A.62}$$

$$AMsz = 1.0 \tag{A.63}$$

$$AMsrol = 0.25 \tag{A.64}$$

$$AMspit = 1.0 \tag{A.65}$$

$$AMsyaw = 0.6 \tag{A.66}$$

Where AMsx is the added mass factor in surge, AMsy is the added mass factor in sway, AMsz is the added mass factor in heave, AMsrol is the added mass factor in roll, AMspit is the added mass factor in pitch and AMsyaw is the added mass factor in yaw.

The mass radius of gyration (squared) are:

$$rxs^2 = 0.003L^2 \tag{A.67}$$

$$rys^2 = 0.05L^2 \tag{A.68}$$

$$rzs^2 = 0.06L^2 \tag{A.69}$$

Where $L$ is the length of the ship.

The effective mass of the ship can be expressed as:

$$Mse = Ms/Cos \tag{A.70}$$

$$Ms = shipmass \tag{A.71}$$

Where $Cos = lsb^2/(1+AMsx)+msb^2/(1+AMsy)+nsb^2/(1+AMsz)+\lambda ls^2/((1+AMsrol)rxs^2) + \mu ls^2/((1+AMspit)rys^2) + \eta ls^2/((1+AMsyaw)rzs^2)$.

Using the added mass assumptions, this simplifies to:

$$Cos = lsb^2 + msb^2/1.5 + nsb^2/2 + \lambda ls^2/(1.25rxs^2) + \mu ls^2/(2rys^2) + \eta ls^2/(1.6rzs^2) \tag{A.72}$$

### A.1.7 Description of the Mass Reduction Coefficient Co for an Ice Floe in 2.5D Simulation

At any point (xrf,yrf) on a 'sloped' ice impact ($\beta > 0$) the inward normal vector can be expressed as:

$$\mathbf{Nfb} = lfbi + mfbj + nfbk \tag{A.73}$$

$$lfb = lrf\cos(\beta) \tag{A.74}$$

$$mfb = mrf\cos(\beta) \tag{A.75}$$

$$nfb = sin(\beta) \tag{A.76}$$

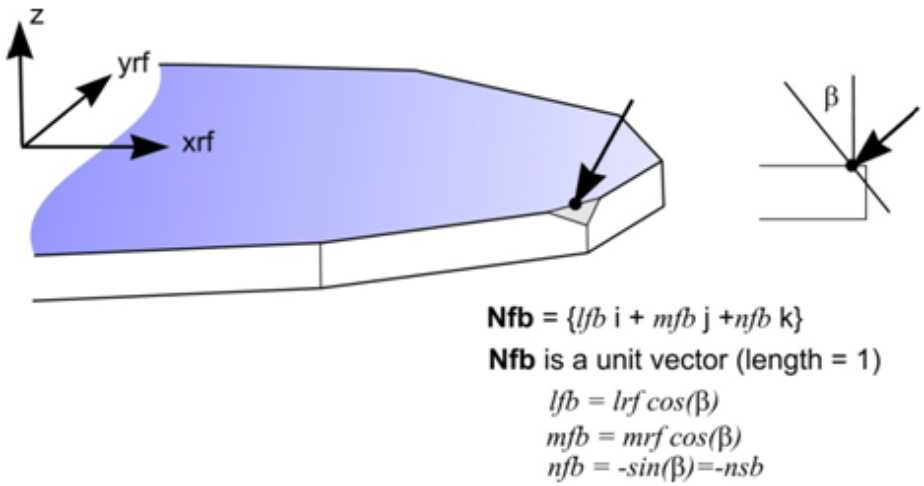



Figure A.5: 2.5D ice floe collision point geometry.

And the moment arms are (assume z=0):

$$\lambda lf = nfbyrf - mfbz \tag{A.77}$$

$$\mu lf = lfbz - nfbxrf \tag{A.78}$$

$$\eta lf = mfbxrf - lfbyrf \tag{A.79}$$

The added mass terms are as follows (estimates):

$$AMfx = 0 \tag{A.80}$$

$$AMfy = 0 \tag{A.81}$$

$$AMfz = 1.0 \tag{A.82}$$

$$AMfrol = 1.0 \tag{A.83}$$

$$AMfpit = 1.0 \tag{A.84}$$

$$AMfyaw = 0 \tag{A.85}$$

Where AMfx is the added mass factor in surge, AMfy is the added mass factor in sway, AMfz is the added mass factor in heave, AMfrol is the added mass factor in roll, AMfpit is the added mass factor in pitch and AMfyaw is the added mass factor in yaw.

The mass radius of gyration (squared) are:

$$rxf^2 = Ixx/A \tag{A.86}$$

$$ryf^2 = Iyy/A \tag{A.87}$$

$$rzf^2 = Izz/A \tag{A.88}$$

Where $Ixx, Iyy, Izz$ are the principal area moments. $A$ is the area of the floe.

The effective mass of the ice floe can be expressed as:

$$Mfe = Mf/Cof \tag{A.89}$$

$$Mf = \rho At = 900At \tag{A.90}$$

Where $Cof = lfb^2/(1 + AMfx) + mfb^2/(1 + AMfy) + nfb^2/(1 + AMfz) + \lambda lf^2/((1 + AMfrol)rxf^2) + \mu lf^2/((1 + AMfpit)ryf^2) + \eta lf^2/((1 + AMfyaw)rzf^2)$.

Using the added mass assumptions, this simplifies to:

$$Cof = lfb^2 + mfb^2 + nfb^2/2 + \lambda lf^2/(2rxf^2) + \mu lf^2/(2ryf^2) + \eta lf^2/(rzf^2) \tag{A.91}$$

### A.1.8 Description of the Ship-Ice Impact Calculation

The normal force can be found by:

$$F_n = P_0.fa.(\frac{KE_e.fx}{P_0.fa})^{\frac{fx-1}{fx}} \tag{A.92}$$

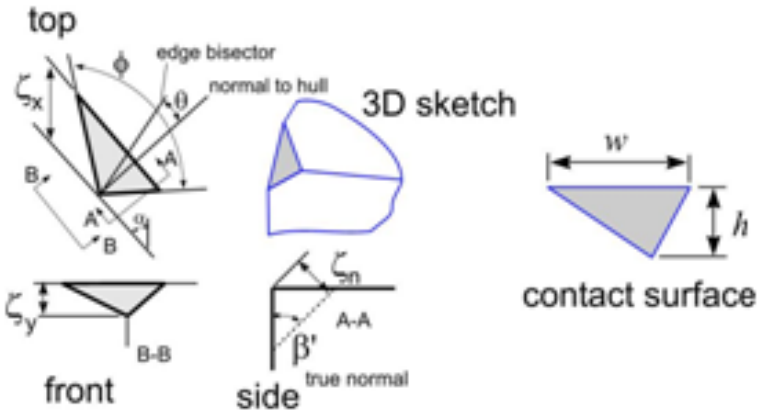



Figure A.6: Ship-ice mpact.

Where $KE_e = \frac{Me}{2}.Vnb^2$, $Vnb = Vn\cos(\beta)$, $Vn$ : 2-body closing speed along 2D normal and $Me = \frac{1}{1/Mse+1/Mfe}$.

For initial simplicity:

$$P_0 = 1MPa \tag{A.93}$$

$$fx = 3 \tag{A.94}$$

$$ex = 0 \tag{A.95}$$

$$F_n = 1.3104P_0^{1/3}.fa^{1/3}.Me^{2/3}.Vnb^{4/3} \tag{A.96}$$

$fa$ in (deg):

$$fa^{1/3} = (0.012\phi + 0.6)(1.38 - 0.027\beta + 0.0004\beta^2) \tag{A.97}$$

$fa$ in (rad):

$$fa^{1/3} = (0.69\phi + 0.6)(1.38 - 1.55\beta + 1.31\beta^2) \tag{A.98}$$

The limit on the normal force can be found by:

$$F_{nlim} = \frac{0.382\sigma h^2 \phi K_v}{(\sin(\beta) - \mu\cos(\beta)) - 0.039(\cos(\beta) + \mu\sin(\beta))} \tag{A.99}$$

Where $K_v = Max(1, 2.14FR^{0.33})$, $FR = \frac{Vnb}{\sqrt{gh}}$, $\sigma = 5$ x $10^5$ N/m$^2$, $h$ is the thickness of the ice, $\mu$ is the friction factor and $g = 9.81$ m/$s^2$.

The ice floe will break into two floes if the normal force greater than the limit on the normal force ($F_n > F_{nlim}$). The length of the break can be found as:

$$L_{cusp} = 1.2min((20 * h/K_v), L_c) \tag{A.100}$$

Where $L_c$ is the distance from the point of contact to the centroid of the floe.

The velocites of the two new floes can be found as:

$$\vec{V_1} = \vec{\omega} x C\vec{C}_1 + \vec{V} \tag{A.101}$$

$$\vec{V_2} = \vec{\omega} x C\vec{C}_2 + \vec{V} \tag{A.102}$$

$$\vec{\omega_1} = \vec{\omega} \tag{A.103}$$

$$\vec{\omega_2} = \vec{\omega} \tag{A.104}$$

Where $\vec{V}$ is the linear velocity of the original floe, $\vec{V_1}$ is the linear velocity of the first new floe, $\vec{V_2}$ is the linear velocity of the second new floe, $\vec{\omega}$ is the angular velocity of the original floe, $\vec{\omega_1}$ is the angular velocity of the first new floe, $\vec{\omega_1}$ is the angular velocity of the second new floe, $C\vec{C}_1$ is the distance from the centroid of the first new floe to the centroid of the original floe and $C\vec{C}_2$ is the distance from the centroid of the second new floe to the centroid of the original floe.

### A.1.9 Description of the Water Drag

The change in the velocities of the ice floes from the water drag force can be found as:

$$dVx = -\frac{1}{2}\frac{\rho_{water} Vx^2 C_d \Delta t}{\rho_{ice} t_{ice}} \tag{A.105}$$

$$dVy = -\frac{1}{2}\frac{\rho_{water} Vy^2 C_d \Delta t}{\rho_{ice} t_{ice}} \tag{A.106}$$

$$d\omega = -\frac{1}{2}\frac{\rho_{water} \omega^2 C_d \Delta t}{\rho_{ice} t_{ice}} \tag{A.107}$$

Where $Vx$ is the current linear velocity of the ice floe in the x-direction, $Vy$ is the current linear velocity of the ice floe in the y-direction, $\omega$ is the current angular velocity of the ice floe, $t_{ice}$ is the thickness of the ice, $\rho_{water}$ is the water density = 1025 kg/$m^3$, $\rho_{ice}$ is the ice density = 900 kg/$m^3$, $C_d$ is the water drag coefficient = 0.05 and $\Delta t$ is the time step.

### A.1.10 Description of the Current and Wind Force

The current and wind are applied as a force per unit area. The change in the velocities of the ice floes from the current and wind can be found as:

$$\Delta V_{wind} = \frac{F_{wind} \Delta t}{\rho_{ice} t_{ice}} \tag{A.108}$$

$$\Delta V_{current} = \frac{F_{current} \Delta t}{\rho_{ice} t_{ice}} \tag{A.109}$$

$$dVx_{wind} = \Delta V_{wind} \cos(\theta) \tag{A.110}$$

$$dVy_{wind} = \Delta V_{wind} \sin(\theta) \tag{A.111}$$

$$dVx_{current} = \Delta V_{current} \cos(\theta) \tag{A.112}$$

$$dVy_{current} = \Delta V_{current} \sin(\theta) \tag{A.113}$$

Where $F_{wind}$ is the magnitude of the wind force per unit area, $F_{current}$ is the magnitude of the current force per unit area, $\Delta V_{wind}$ is the magnitude of the change of the velocity from the wind force, $\Delta V_{current}$ is the magnitude of the change of the velocity from the current force and $\theta$ is the wind or current direction.

# Appendix B

# Appendix

## B.1 Simulation File Structure

This file (.ice) used to store the simulation data. The file is written in XML (eXtensible Markup Language)[1] and follow a unique scheme created specifically for simulation files. The XML parser CMarkup[2] is used for reading and writing .ice files.

- Each file contains an entire simulation
- The initial frame (index = 0) contains all objects in a simulation
- Subsequent frames only contain updated object information if it changes from the previous frame. This eliminates the unnecessary repetition of object data every frame

[1] http://www.w3schools.com/xml/xml_whatis.asp
[2] http://www.firstobject.com

- The $\langle IceFloeSim \rangle$ element contains the entire simulation which is made up of $\langle frame \rangle$ elements. The $\langle IceFloeSim \rangle$ element has an "numFrames" attribute which represents the number of frames/scenes in the simulation
- Each $\langle frame \rangle$ element represents an individual frame/scene. The frame number is given by the "index" attribute
- Each frame contains $\langle object \rangle$ elements representing an individual object within a frame
- Every object has an "index" attribute which is constant throughout the entire simulation and used to track changes to an object's parameters between frames
- Objects contain parameter elements (currently only $\langle coordinates \rangle$, $\langle velocity \rangle$ and $\langle thickness \rangle$) which describe the object

### B.1.1 Example .ice File

Listing B.1: Simulation File Structure (.ice)

```
1 <iceFloeSim numFrames="INT">
2 |
3 |     <frame index="INT">
4 |   |
5 |   |   <object index="INT">
6 |   |   |  |
7 |   |   |  |  <coordinates> FLOAT FLOAT ... </coordinates>
8 |   |   |  |
```

```
9  |    |      |   |  <velocity x="FLOAT" y="FLOAT" a="FLOAT" />
10 |    |      |   |
11 |    |    | | <thickness> FLOAT </thickness>
12 |    |      | |
13 |    |    </object>
14 |    |
15 |    |    .
16 |    |      .
17 |    |      .
18 |    |
19 |    |
20 |   </frame>
21 |
22 |       .
23 |       .
24 |       .
25 |
26 </iceFloeSim>
```

## B.2 Simulator Design

Table B.1 describes the classes that are used in the ice simulator.

| Class Name | Description |
|---|---|
| FileParser | Reads the input file and exctracts the information about polygons |
| Frame | Stores all required information about the scene |

| | |
|---|---|
| Markup | Interface for the XML parser |
| Point | Stores the coordinates for vertices of the polygon |
| PolygonObj | Stores all information about the polygon and provides several functions that can be used to calculate the properties (mass, area, centroid, etc...) of the polygon |
| Polygon_Kernel | Contains the computationally intensive work which is implemented as kernels and executed on the GPU |
| PolygonSystem | C wrapper class |
| Simulation | Stores the simulation data |
| Simulator | Provides functions that can be used to run and control the simulation |

Table B.1: The classes in the ice simulator

# References


[1] S. A., B. Leira, , and K. Riska. Short term extreme statistics of local ice loads on ship hulls. *Cold Regions Science and Technology*, pages 130–143, 2012.

[2] J. Adams, J. Sheppard, S. Alawneh, and D. Peters. Ice-floe simulation viewer tool. In *Proceedings of Newfoundland Electrical and Computer Engineering Conference (NECEC 2011), IEEE*, St. John's, NL, Canada, Nov 2011.

[3] B. Barney. Introduction to parallel computing. Technical report, Lawrence Livermore National Laboratory, 2013. https://computing.llnl.gov/tutorials/parallel_comp.

[4] J. Blanchette and M. Summerfield. *C++ GUI Programming with Qt 4 (2nd Edition) (Prentice Hall Open Source Software Development Series)*. Prentice Hall, 2 edition, Feb. 2008.

[5] J. Bolz, I. Farmer, E. Grinspun, and P. Schröoder. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. *ACM Trans. Graph.*, 22(3):917–924, July 2003.

[6] A. H. Bond. Havok fx: GPU-accelerated physics for pc games. In *Proceedings of Game Developers Conference 2006*, mar 2006. http://www.havok. com/content/view/187/77/.

[7] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. Brook for GPUs: stream computing on graphics hardware. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 777–786, New York, NY, USA, 2004. ACM Press.

[8] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, Aug. 1991.

[9] C. Daley. Energy based ice collision forces. In *POAC '99*, Helsinki, Finland, 1999.

[10] C. Daley, S. Alawneh, D. Peters, and B. Colbourne. GPU-event-mechanics evaluation of ice impact load statistics. In *Offshore Technology Conference (OTC 2014)*, Houston, Texas, USA, 2014.

[11] C. Daley, S. Alawneh, D. Peters, B. Quinton, and B. Colbourne. GPU modeling of ship operations in pack ice. In *International Conference and Exhibition on Performance of Ships and Structures in Ice (ICETECH 2012)*, Banff, Alberta, Canada, 2012.

[12] C. Daley and A. Kendrick. Direct design of large ice class ships with emphasis on the midbody ice belt. In *Proc. 27th Int'l Conf. on Offshore Mechanics and Arctic Engineering OMAE2008*, Estoril, Portugal, 2008.

[13] C. Daley, J. St. John, R. Brown, J. Meyer, and I. Glen. Ice loads and ship response to ice - a second season. Technical Report SSC-339, Report to U.S. Ship Structures Committee, 1990.

[14] C. Daley, J. St. John, R. Brown, J. Meyer, and I. Glen. Ice loads and ship response to ice - consolidation. Technical Report SSC-340, Report to U.S. Ship Structures Committee, 1990.

[15] C. Daley, J. St. John, F. Siebold, and I. Bayly. Analysis of extreme ice loads measured on uscgc polar sea. In *Transactions, SNAME*, New York, USA, 1984.

[16] R. Dragt, S. Bruneau, and S. Alawneh. Design and execution of model experiments to validate numerical modelling of 2d ship operations in pack ice. In *Proceedings of Newfoundland Electrical and Computer Engineering Conference (NECEC 2012), IEEE*, St. John's, NL, Canada, Nov 2012.

[17] D. Eberly. Intersection of convex objects: The method of separating axes. *Geometric Tools, LL*, 2008.

[18] D. Eberly. Triangulation by ear clipping. Technical report, Geometrictools, March 2008. http://www.geometrictools.com.

[19] E. Enkvist, P. Varsta, and K. Riska. The ship-ice interaction. In *Proceedings of The International Conference on Port and Ocean Engineering under Arctic Conditions (POAC)*, volume 2, page 9771002, Trondheim, 1979.

[20] C. Ericson. *Real-Time Collision Detection*. Morgan Kaufmann, 2005.

[21] N. Goodnight, C. Woolley, G. Lewin, D. Luebke, and G. Humphreys. A multigrid solver for boundary value problems using programmable graphics hardware. In

*Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, HWWS '03, pages 102–111, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[22] S. Green. Nvidia particle system sample. Technical report, Nvidia, 2004. http://download.developer.nvidia.com/developer/SDK/.

[23] S. Green and M. J. Harris. Game physics simulation on nvidia GPUs. In *Proceedings of Game Developers Conference 2006*, mar 2006. http://www. havok.com/content/view/187/77/.

[24] T. R. Hagen and J. R. Natvig. Solving the Euler Equations on Graphics Processing Units. *Comp. Sci. - ICCS*, pages 220–227, 2006.

[25] M. Harris. Fast fluid dynamics simulation on the GPU. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[26] M. Harris. GPGPU website. Technical report, GPGPU, 2008. http://www.gpgpu.org/.

[27] M. J. Harris, W. V. Baxter, T. Scheuermann, and A. Lastra. Simulation of cloud dynamics on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, HWWS '03, pages 92–101, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[28] Haxon. Ice floe at oslofjord. Technical report, panoramio, mar 2009. http://www.panoramio.com/photo/19618780.

[29] A. Keinonen, B. R. Revill, and A. Reynolds. Ice breaker characteristics synthesis. Technical Report TP 12812 E., Report of AKAC Inc. to Transportation Development Centre, 1996.

[30] P. Kipfer, M. Segal, and R. Westermann. UberFlow: a GPU-based particle engine. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 115–122, New York, NY, USA, 2004. ACM.

[31] A. Kolb, L. Latta, and C. Rezk-Salama. Hardware-based simulation and collision detection for large particle systems. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, HWWS '04, pages 123–131, New York, NY, USA, 2004. ACM.

[32] T. Kotras, A. Baird, and J. Naegle. Predicting ship performance in level ice. *SNAME Transactions 91*, page 329349, 1983.

[33] J. Krüger and R. Westermann. Linear algebra operators for GPU implementation of numerical algorithms. *ACM Trans. Graph.*, 22(3):908–916, July 2003.

[34] P. Kujala. On the statistics of ice loads on ship hull in the baltic. *Mechanical Engineering Series*, pages 1–19, 1994.

[35] P. Kujala. Semi-empirical evaluation of long term ice loads on a ship hull. *Marine Structures*, 9(9):849–871, 1996.

[36] B. Leira, L. Børsheim, ø. Espeland, and J. Amdahl. Ice-load estimation for a ship hull based on continuous response monitoring. *Journal of Engineering for the Maritime Environment*, pages 529–540, 2009.

[37] B. Leira, B.J. Estimation of ice loads on a ship hull based on strain measurements. In *Proceedings of the 27th International Conference on Offshore Mechanics and Arctic Engineering*, Estoril, Portugal, 2008.

[38] G. Lindqvist. A straightforward method for calculation of ice resistance of ships. In *Proceedings of The International Conference on Port and Ocean Engineering under Arctic Conditions (POAC)*, volume 2, page 722735, Lule, 1989.

[39] R. Lubbad and S. Løset. A numerical model for real-time simulation of shipice interaction. *Cold Regions Science and Technology*, 65(2):111 – 127, 2011.

[40] S. Luding. Introduction to discrete element methods : basic of contact force models and how to perform the micro-macro transition to continuum theory. *European Journal of Environmental and Civil Engineering*, 12(7-8):785–826, 2008.

[41] G. H. Meisters. Polygons Have Ears. *The American Mathematical Monthly*, 82(6):648–651, 1975.

[42] D. Muggeridge and A. F. Aboulazm. Analytical investigation of ship resistance in broken or pack ice. In *Proc. 8th Int. Conference, Offshore Mechanics & Arctic Engng*, volume IV, page 359365, 1989.

[43] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with cuda. *Queue*, 6:40–53, March 2008.

[44] Nivida. Physx. Technical report, Nvidia, 2004. http://www.geforce.com/hardware/technology/physx.

[45] Nvidia. Cuda architecture overview v1.1. introduction & overview. Technical report, Nvidia, 2009.

[46] Nvidia. Cuda development tools v2.3. getting started. Technical report, Nvidia, 2009.

[47] Nvidia. Cuda programming guide v2.3.1. Technical report, Nvidia, 2009.

[48] Nvidia. Tesla c2050. Technical report, Nvidia, 2010. http://www.nvidia.com/object/tesla-supercomputing-solutions.html.

[49] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, New York, NY, USA, 2nd edition, 1998.

[50] J. Owens. Streaming architectures and technology trends. In M. Pharr, editor, *GPU Gems 2*, chapter 29, pages 457–470. Addison Wesley, Mar. 2005.

[51] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.

[52] V. Pascucci. Isosurface computation made simple. In *VisSym*, pages 293–300, 2004.

[53] Y. Popov, O. Faddeyev, D. Kheisin, and A. Yalovlev. Strength of ships sailing in ice. *Sudostroenie Publishing House, Leningrad*, 1967.

[54] A. Rich, J. St. John, and R. Browne. Ice load impact measurements on the ccgs louis s. st.-laurent during the 1994 arctic ocean crossing - analysis and conclusions. Technical report, Report to Canadian Coast Guard and US Coast Guard, 1997.

[55] K. Riska, M. Patey, S. Kishi, and K. Kamesaki. Influence of ice conditions on ship transit times in ice. In *Proceedings of The International Conference on Port and Ocean Engineering under Arctic Conditions (POAC)*, volume 2, page 729745, Ottawa, Ontario, Canada, 2001.

[56] N. Satish, M. Harris, and M. Garland. Designing efficient sorting algorithms for manycore GPUs. In *Proceedings of the 2009 IEEE International Symposium on Parallel and Distributed Processing*, IPDPS '09, pages 1–10, Washington, DC, USA, 2009. IEEE Computer Society.

[57] M. I. Shamos. *Computational geometry*. PHD thesis, Yale University, New Haven, 1978.

[58] J. St. John, C. Daley, and H. Blount. Ice loads and ship response to ice. Technical Report SSC-329, Report to U.S. Ship Structures Committee, 1985.

[59] N. Stewart, G. Leach, and S. John. Improved CSG rendering using overlap graph subtraction sequences. In *Proceedings of the International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia (GRAPHITE 2003)*, pages 47–53. Acm Press, 2003.

[60] M. Suominen and P. Kujala. Analysis of short-term ice load measurements on board ms kemira during the winters 1987 and 1988. Technical Report AM-22, Report to Aalto University, School of Science and Technology, Department of Applied Mechanics. Espoo, Finland, 2010.

[61] D. Tskhakaya. The particle-in-cell method. In H. Fehske, R. Schneider, and A. Weie, editors, *Computational Many-Particle Physics*, volume 739 of *Lecture Notes in Physics*, pages 161–189. Springer Berlin Heidelberg, 2008.

[62] P. Valanto. The resistance of ships in level ice. *SNAME Transactions 109*, page 5383, 2001.

[63] R. C. Woolgar and D. B. Colbourne. Effects of hullice friction coefficient on predictions of pack ice forces for moored offshore vessels. *Ocean Engineering*, 37(23):296 – 303, 2010.

[64] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu. *The Finite Element Method: Its Basis and Fundamentals, Sixth Edition*. Butterworth-Heinemann, 6 edition, may 2005.