Hyper-Real-Time Ice Simulation and Modeling Using GPGPU

Shadi Alawneh, *Young Professional Member, IEEE,* Roelof Dragt,
Dennis Peters, *Senior Member, IEEE,* Claude Daley, and Stephen Bruneau

Abstract—This paper describes the design of an efficient parallel implementation of an ice simulator that simulates the behaviour of a ship operating in pack ice. The main idea of the method is to treat ice as a set of discrete objects with very simple 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 simulator is developed using the NVIDIA Compute Unified Device Architecture (CUDA). This paper 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. Our results show speed up of 11 times, reducing simulation time for a large ice field (9801 floes) from over 2 hours to about 12 minutes.

Index Terms—GPGPU, CUDA, Ice Simulation, Ice - Ship Interaction, Experimental Validation.

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 paper 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 can destroy 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 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 (Graphical Processing Unit) hardware. The main idea of the method is to treat ice as a set of discrete objects with very simple 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) methods, 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). Unlike existing methods such as finite element [1] discrete element [2] and Particle in Cell [3] methods are built on the ideas of continuum mechanics. The IEMM approach is based on the premise that aggregate behavior is only weakly dependent on the continuum processes inside ice events, but very strongly dependent on the sequence of events. Each discrete collision and failure (fracture) that occurs creates the initial conditions for the subsequent event. The collisions and fractures occur so fast (relative to the time between events) that they can be considered to be instant, which is to say that they are events rather than processes.

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 behavior. The approach permits the development of useful solutions to a number of practical problems that have been

- *S. Alawneh, D. Peters, C. Daley and S. Bruneau are with the Faculty of Engineering, Memorial University of Newfoundland, St. John's, NL, A1B 3X5.*
  E-mail:{shadi.alawneh,dpeters,cdaley,sbruneau}@mun.ca
- *R. Dragt is with Delft University of Technology.*
  E-mail: sanderdragt@gmail.com

This paper was presented in part at the 14th IEEE International Conference on High Performance Computing and Communications, 2012

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

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 will enable 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.

This paper presents an efficient parallel implementation of such a simulator developed using the NVIDIA Compute Unified Device Architecture (CUDA). This paper also presents the results of the experiments to evaluate the performance of the algorithms developed in this work and to validate the numerical models of ship operations 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) as they move under the influence of currents and wind and interact with land, ships and other structures, while 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 as such it is necessary to achieve a simulation many times faster than real-time simulation.

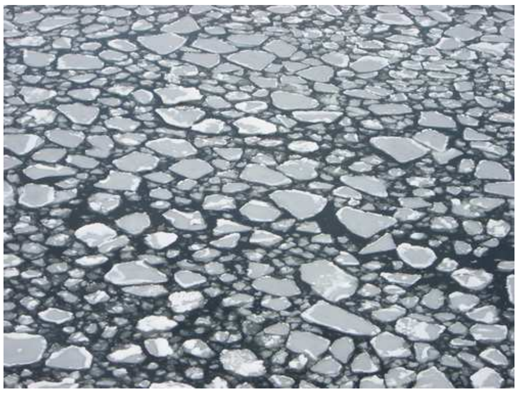
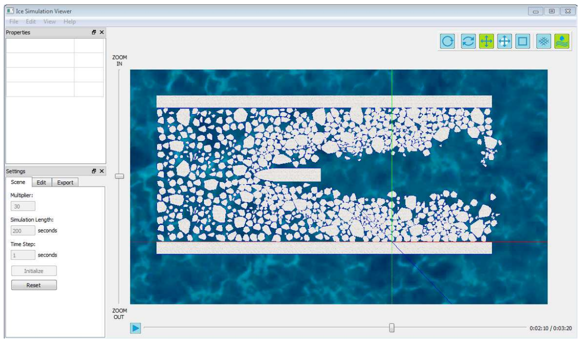


Fig. 1. Ice floes.[4]

This project is structured in two components, the *Ice Simulation Engine*, which is the focus of this paper, and the *Ice Simulation Viewer*, which is being developed to display 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 [5] user interface framework. Figure 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 [6].



Multiplier: 30
Simulation Length: 200 seconds
Time Step: 1 seconds
0:02:10 / 0:03:20

Fig. 2. Ice simulation viewer

This paper handles the 2D simulation of pack ice and consider driving forces (e.g., current, wind) and models some 3D aspects but didn't consider motion in 3D. The goal is to achieve a simulation that is fast enough to be practically used for planning ice management activities in realistic size ice fields. The 3D version of the simulation will be left for future work.

1.2 Novel Contributions

Through our literature review we were not able to find any other published work in which GPGPU is used to achieve hyper-real-time simulations of ice. Therefore, this paper introduces a new GPGPU approach that can be used to simulate the behaviour of a ship in pack ice. Using the new GPGPU approach, 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. This approach will be described in detail in Section 5.

1.3 Paper Outline

The remainder of this paper is organized as follows. Section 2 presents the related work. Section 3 describes the mechanics of the GPGPU model. Section 4 describes a brief overview of CUDA and it also describes the collision detection algorithm and a high level algorithm of the simulator. Section 5 describes the experiments to evaluate the performance of algorithms and it also discusses the different approaches for implementing the ice simulator. Section 6 describes the physical model experiments to validate the GPGPU model. Sections 7 and 8 concludes with the ongoing research and future plans.

2 Related Work

The work reported in this paper is an extension of the recent work of [7], [8]. The experiments in the current work shows the performance for a larger ice fields. The current work introduces a new approach to generate the list of neighbors for each ice floe, which results in better performance. This approach uses a variable radius specific to each floe pair. The current work also discusses the performance of alternative collision detection approach (uniform grid) and describes the execution of the experiments to validate the numerical modeling of ship operations in pack ice. The validation experiments has been described in a non-refereed local workshop [9].

A complete details about the Ice Simulation Viewer, which is being developed to display the data produced by the simulation engine, has been described in a non-refereed local workshop [6].

The event-mechanics approach to assess vessel performance in pack ice has been introduced in [10], [11]. In [10], 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. In [11], the use of a GPU-Event-Mechanics (GEM) simulation to assess local ice loads on a vessel operating in pack ice has been explored.

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 [12]. Kotras et al. [13] and Valanto [14] 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 breaking.

A good understanding of the processes of ship-ice interaction is essential for developing reliable theoretical models. These models help optimize the design and operation of ships in Arctic waters. Several performance models exist, including semi-analytical and purely empirical variants, e.g. [15], [16], [17]. 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 optimize their route.

Lubbad et al. [18] described a numerical model to simulate the process of ship-ice interaction in real-time. PhysX, a real-time physics engine middleware SDK, is used 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 our 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, so it is not used in this project.

There are several researchers who have developed particle system simulation on GPUs. Kipfer et al. [19] 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. [20] 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 [21]. They described how to implement a particle system in CUDA, including particle collisions using a uniform grid data structure which will be described in section 5.3.1. In this work, we have tried to use the uniform grid data structure to handle collisions but we didn't get better performance than the current approach that we are using.

3 Ice Behaviour

Each ice-ice collision event within the pack is treated using a method that can be traced to Popov et. al [22]. The method was updated to reflect pressure-area effects [23], and used for a variety of ship-ice interaction scenarios [24]. 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). 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 4). 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.

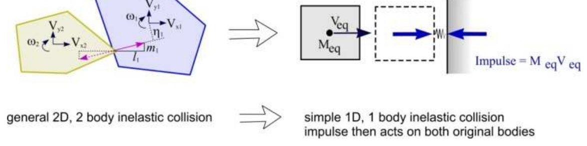

V_{y1}
ω_1
V_{x1}
η_1
m_1
l_1
V_{y2}
ω_2
V_{x2}
V_{eq}
M_{eq}
Impulse = $M_{eq}V_{eq}$
general 2D, 2 body inelastic collision
simple 1D, 1 body inelastic collision
impulse then acts on both original bodies

Fig. 3. Idealization of 2D collision between two finite bodies. [11]

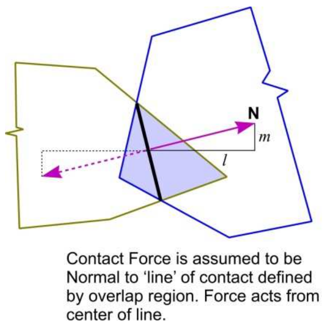

N
m
l
Contact Force is assumed to be Normal to 'line' of contact defined by overlap region. Force acts from center of line.

Fig. 4. Assumption concerning the location and direction of impact forces. [11]

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

$$R = l^2 + m^2 + \frac{\eta^2}{r_x^2} \quad (1)$$

Where l and m are direction cosines of the inward normal vector, η 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). Each body in a two body collision has a unique mass reduction factor. The above mass reduction factor represents the simplest case 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 [23]).

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 [23]). 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.

4 METHODOLOGY

4.1 CUDA Overview

CUDA is a comprehensive software and hardware architecture for GPGPU that was developed and released by Nvidia in 2007. This development forwarded Nvidia's move toward GPGPU and High-Performance Computing (HPC), combining huge 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 [25].

Based on [26], the CUDA programming model provides 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.

4.2 Collision Detection

Since this work uses a discrete time simulation, for each time step the collisions are detected by searching for regions of overlap between ice floes, compute the momentum that would result from such a collision and adjust 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 we have used the method of separating axes [27]. This method determines whether or not two 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 Fig. 5 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).

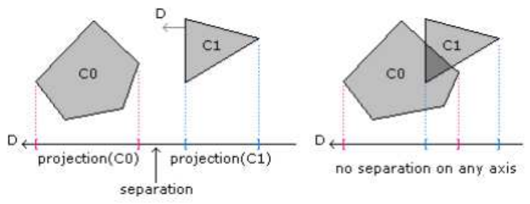


Fig. 5. Nonintersecting convex polygons (left). Intersecting convex polygons (right).[27]

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 arbitrary polygons of n and m vertices can 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 [28], and since then a variety of different algorithms have been developed, all achieving $O(n + m)$ time complexity. This work uses the algorithm that was developed by O'Rourke, Chien, Olson & Naddor [29].

The basic idea of the algorithm is as illustrated in Algorithm 1[29]. 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. The edges A and B are shown as vectors.

4.3 High Level Algorithm of The Simulator

Fig. 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

Algorithm 1 :Intersection of convex polygons

```
                                   ▷ Assume 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 geometric conditions.
until both $A$ and $B$ cycle their polygons
if no intersections were found then
    One polygon must be entirely within the other.
end if
```

work of the simulation. First, the "create neighbours list" kernel is launched to find the list of polygons 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 floe data is transferred back to the CPU. This process is repeated until the simulation is completed.

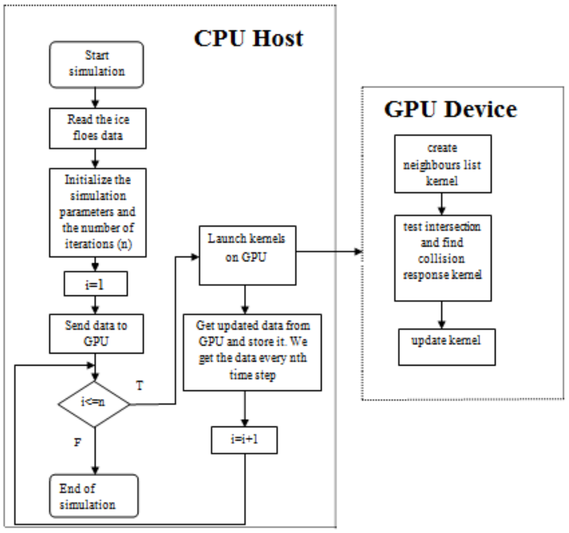

CPU Host
Start simulation
Read the ice floes data
Initialize the simulation parameters and the number of iterations (n)
i=1
Send data to GPU
i<=n
T
F
End of simulation
Launch kernels on GPU
Get updated data from GPU and store it. We get the data every nth time step
i=i+1
GPU Device
create neighbours list kernel
test intersection and find collision response kernel
update kernel

Fig. 6. Ice simulator flowchart.

5 Algorithms Development

This section describes the experiments to evaluate the performance of algorithms. It also discusses the different approaches for implementing the ice simulator.

In this work, Intel(R) Xeon(R) CPU E5520 @2.27GHz and a GPU Tesla C2050 card have been used. This card has 448 processor cores, 1.15 GHz processor core clock and 144 GB/sec memory bandwidth.

5.1 Ice Simulator Implementation

As the 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 7.

In the first implementation, two CUDA kernels were used: The first kernel, executed using one thread per polygon, finds the list of all pair-wise collisions by determining which pairs of polygons (ice floes) are overlapping. The second kernel, executed using one thread per overlapping polygon pair, 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, and didn't achieve real-time results in all cases and therefore is insufficient.

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

In the third implementation we took advantage of the fact that polygons that are widely separated are unlikely to overlap any time soon, and so we could dramatically reduced the number of polygons to be checked for collision by eliminating those that are beyond some distance away. To do this we added another kernel that finds the list of neighbours for each polygon that are within the region where they might overlap with it soon. Therefore, instead of checking the collisions with every other object we just checked with those in the list of neighbours. The list is re-created periodically, but not at every time step, so that the total number of computations is significantly reduced. This approach is significantly faster than the other two approaches as we see in Figure 7 and achieves substantially better than real-time simulation for small ice fields.

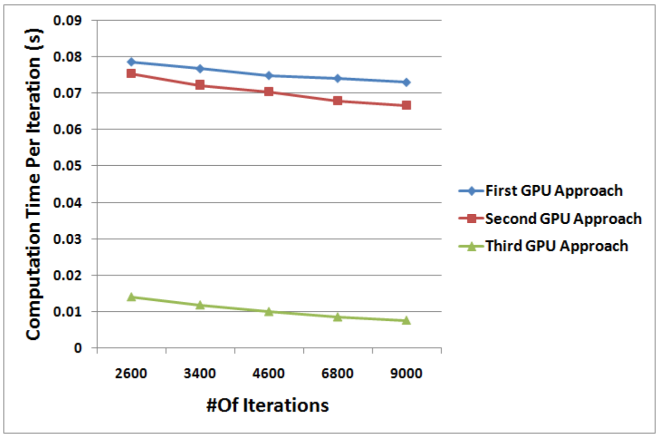


Fig. 7. Computation time per iteration of the three GPU approaches (456 floes).

5.2 Performance Evaluation

We have implemented a serial and parallel version of the simulator and tested both versions on a large ice fields and at different (real-time) durations. The ice field has 9801 ice floes. The simulation time step (Δt) that we have used in the simulations is 0.1s. We have used 0.1s to maintain accuracy in the ice mechanics. We have tried two different approaches to generate the list of neighbors for each ice floe: In the first approach, we have used a fixed radius of 70m around each floe in the entire ice field. This radius is selected based on the maximum velocity that we could have in the ice field. In the second approach, we find the list of neighbours using a variable radius specific to each floe pair. Finally, we have done another experiment on ice fields of different sizes to show the scalability of the parallel implementation.

5.2.1 Results

The CPU computation time per iteration to simulate the behaviour of the ship in a large ice field, which has 9801 ice floes, for all five different durations (4000,5000,6000,7000,8000), is about 1 second but the GPU time is about 0.09 second. Therefore, the speed up of using the GPU approach is about 11 times. Moreover, the simulation is hyper-real-time since the computation time per iteration is less than the simulation time step ($\Delta t = 0.1s$).

Figure 8 shows the CPU and GPU computation time per iteration to simulate the behaviour of the ship in different ice fields, for the same number of iterations (1850). As we see in Figure 8 we can tell that the GPU approach gets faster than the CPU approach as the number of ice floes increases. Moreover, the simulation is hyper-real-time since the computation time per iteration is less than the simulation time step ($\Delta t = 0.1s$).

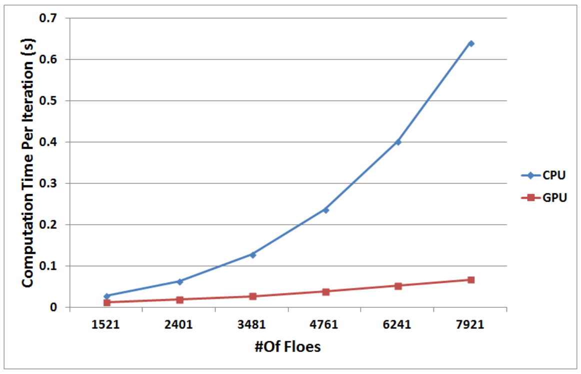


Fig. 8. Computation time per iteration for different ice fields.

Figure 9 shows the speed up of the GPU approach using ice fields of different sizes. As we see in Figure 9 we can tell that the speed up increases as the number of ice floes increases.

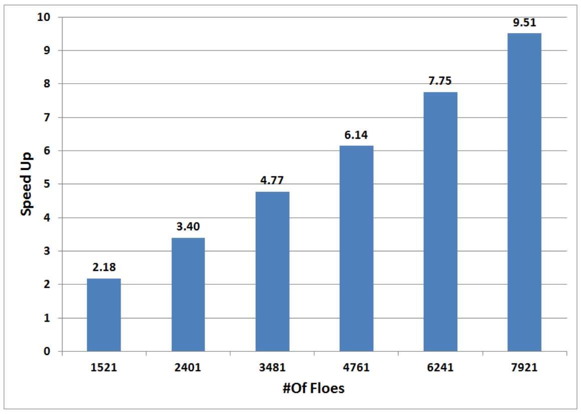


Fig. 9. GPU approach speed up for different ice fields.

5.3 Alternative Collision Detection Approach

This section describes alternative collision detection approach that we have tried. It also discusses the performance evaluation of the approach.

5.3.1 Uniform Grid Data Structure

In the uniform grid approach [30], a grid subdivides the simulation space into a grid of uniformly sized cells. For the sake of simplicity, we have used a grid where the cell size is the same as the size of the largest ice floe (double its radius). Also, the grid that we have used is called a "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 in the collision processing we must also examine the ice floes in the neighboring cells (9 cells in total in 2D grid) 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. We have used the linear cell id (the address calculation is always bounded by the grid size which is a power of two) as the hash. The kernel stores the results to the "particleHash" array in global memory as a pair of unsigned integers (uint2) pair (cell hash, ice floe id).

We then sort the ice floes 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 [31]. This creates a list of ice floe ids in cell order. In order for this sorted list to be useful, we need to be able to find the start of any given cell in the sorted list. This is done by running another kernel "findCellStart", which uses one 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.

5.3.1.1 Results: Figure 10 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 we have discussed in section 5.2 and Uniform Grid is the computation time using the uniform grid approach.

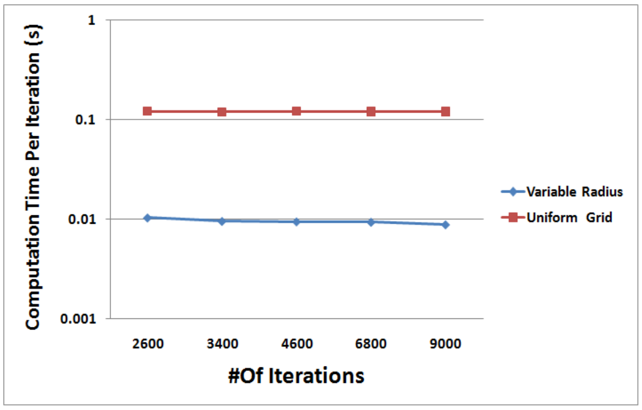


Fig. 10. Computation time per iteration using the uniform grid and list of neighbours approaches (456 floes).

As we see in Figure 10 we can tell that the uniform grid approach is slower than the list of neighbours approach. Therefore, we have used the list of neighbours approach in our implementation.

6 Model Validation

In the model validation we modeled the ship and the ice floes as polypropylene pieces. Then, we did a physical experiments in the tank. After that we compared the velocities and the positions of some of the ice floes and the ship that obtained from the physical experiments with the velocities and the positions of the same ice floes and the ship that obtained from the numerical simulation. In the physical experiments we didn't model the contacts. Also, if there is no ice that means there will be no contact. In the model validation we just validated the positions and velocities of the ship and some of the ice floes. We didn't validate the forces.

6.1 Modelling the GPGPU Model

The GPGPU model is validated using physical model experiments, which are designed to approach the boundary conditions of the GPGPU model as closely as possible. Most important are 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 11 shows the 2D concept and the axis used.

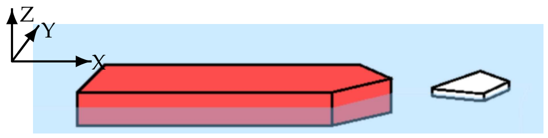


Fig. 11. Schematic view of the 2D concept used in the model.

6.2 Physical Model Experiments

The main goal, as described above is divided into five subgoals:

A To develop a repeatable method of creating ship-floe and floe-floe collisions in the lab.
B To develop a reliable method to register and measure positions and rotations over a given period of time.
C To develop a method to analyze and quantify the results.
D To develop a method to compare the results to a numerical simulation.
E Validate the numerical model and make recommendations.

6.2.1 *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 12.

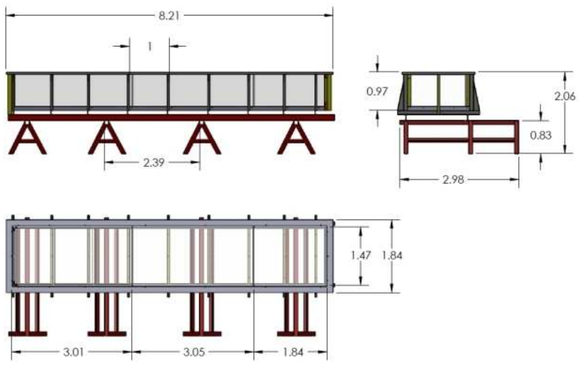


Fig. 12. Drawing of the acrylic tank, dimensions are in meters.

The ship and the floes are constructed out of polypropylene, with a density of 905 kg/m^3, which is the density of ice. Polypropylene is chosen because it has the right density and it doesn't melt. These properties are close to the boundary conditions of the GPGPU model, which assumes rigid body behaviour and completely elastic collisions. 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 inches) and randomly shaped into convex polygons with three to six sides. The vessel itself is made out of a 50.8 mm thick (2 inches) sheet of polypropylene, has a 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 13 shows the vessel with its main dimensions.

The vessel is controlled using an overhead carriage, which is connected to the beam of the vessel using an aluminum rod. 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 14). Unfortunately, the overhead carriage is 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) is not entirely met. However, the error introduced is relatively small, as is shown in subsection 6.2.4.

6.2.2 Method to register and measure positions and rotations over a given period of time

Movements of vessel and the floes in the tank are recorded on camera during the entire experiment. This camera is located under the tank and looks up through the acrylic bottom of the tank. This way, access to the camera is easier, but the camera's view is limited to one segment of the tank, due to the structural support of the acrylic tank. This limited the maximum dimensions of the ice field.

Also, due to the camera placement under the tank, some refraction takes place when the light travels from the water, through the acrylic glass to the air. The theoretical refraction is calculated and the results showed that due to this specific combination of water, acrylic and air only a neglectable refraction occurs.

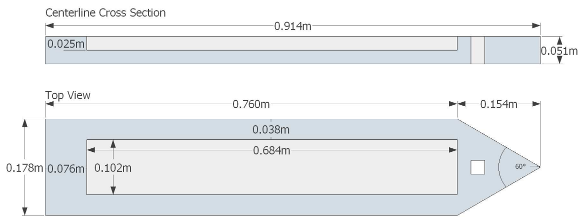


Fig. 13. Design drawing of the vessel used in the experiments.

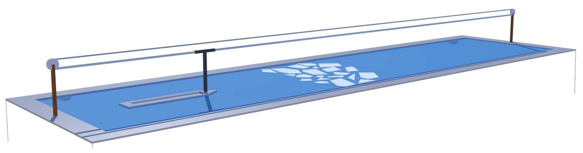


Fig. 14. Schematic experiment layout, showing the vessel, some floes and the towing carriage above the tank.

The floes and ship are outfitted with targets, so they can be tracked using image processing software. A "bow tie" target is designed which differs a little from the typical cross-pattern, as is seen in Figure 15. This design is chosen because it presents a single large coloured surface (a cross-pattern design has two), which makes it easier to find the target and remove computational noise. Also, the design makes it easier to recognize the direction of the floe and thus calculate the rotational velocity during the experiment.

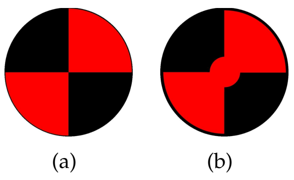


Fig. 15. Cross design (a) and Bow Tie design (b).

6.2.3 Method to analyze and quantify the results

The method used to analyze the data filters all but the given colour range from the frames. The exact location and orientation of the target is then calculated and saved for each frame in the video. The velocities are calculated by comparing the change in target position between frames, with a frame rate of 30 frames per second.

The camera footage, Figure 16 shows one frame, is processed using Matlab. First of all, the user interface enables the user to crop and rotate the video. By doing this, all the parts of the image outside of the tank boundaries are removed and the sides and top of the image can be used as a reference frame to determine the exact location of the floes and the vessel. Also, the user is able to determine which part of the video is processed and which colours are tracked.

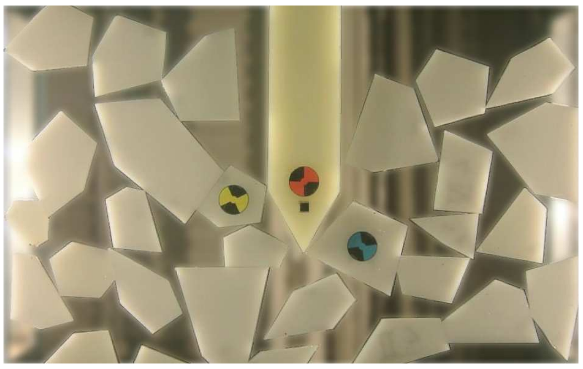


Fig. 16. Bottom-up view of floes and a ship, outfitted with "bow tie" targets.

The processing starts with separating the colour information (rgb or red, green and blue) into separate matrices. Next, a colour threshold is used to find the colour targets. This threshold sets the value of the pixel to 255 if the colour is within the threshold and to 0 if it is not. Combining the information of all three images (red, green and blue) gives Figure 17(a). The white areas are within (all) the thresholds, the coloured images are within one or two rgb values and the black areas are outside the thresholds. This colourful image is used to calibrate the threshold, as the colours show which part of the image has to be altered. Finally the image is converted into a binary image, Figure 17(b), for further processing.

Using the built-in functions *filling* and *area*, the gaps in the "bow tie" are filled and noise (small areas that fall within the threshold) is removed. The result is shown in Figure 17(c). Using this image, the centroid and orientation of the "bow tie" is calculated and saved. Next, the following frame is analyzed in the same way.

The colours most suitable for this tracking method are blue, yellow and red. Green is also usable, although it is found to be quite close to the shade of blue. Using this method, two targets of the same colour cannot be tracked at the same time, because the program will just average the two and place the centroid between targets. However, for the first validation of the model, tracking only four targets (the ship and three floes) suffices.

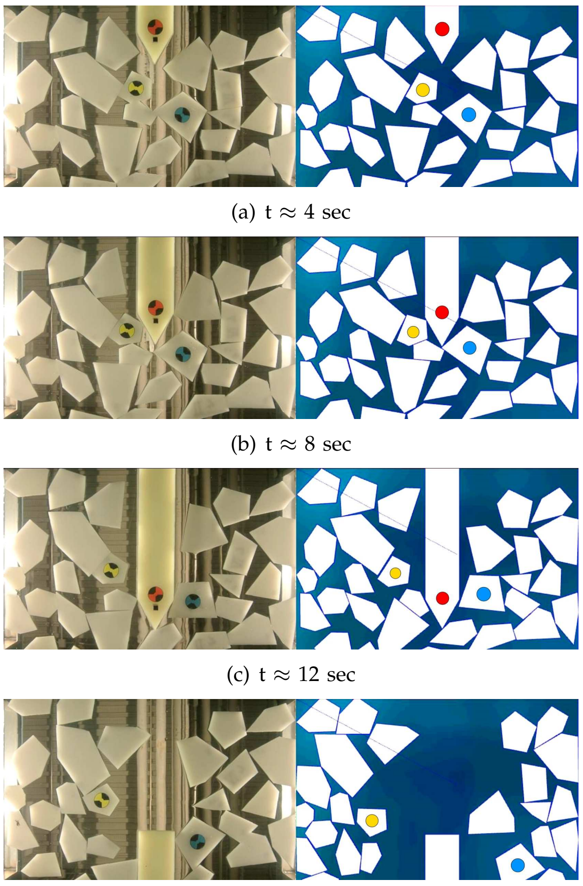
6.2.4 *Method to compare the results to a numerical simulation*

The starting position of all the floes, taken at the time of first ship contact, is manually converted into a 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 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 18 and a pack ice simulation is shown in Figure 19. 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.

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 the comparison with the experiment data by placing both videos next to each other, as is shown for four frames in Figure 20.



(a) t ≈ 4 sec

(b) t ≈ 8 sec

(c) t ≈ 12 sec

(d) t ≈ 25 sec

Fig. 20. Comparison between the numerical simulation and the experiments of a single case. The bodies in the numerical model are manually given coloured dots for convenience.

6.2.5 *Validation of the numerical model 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 insufficient in the GPGPU model. This shows floes (in open water, see Fig. 18) 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 19), shows positions and velocities at the early

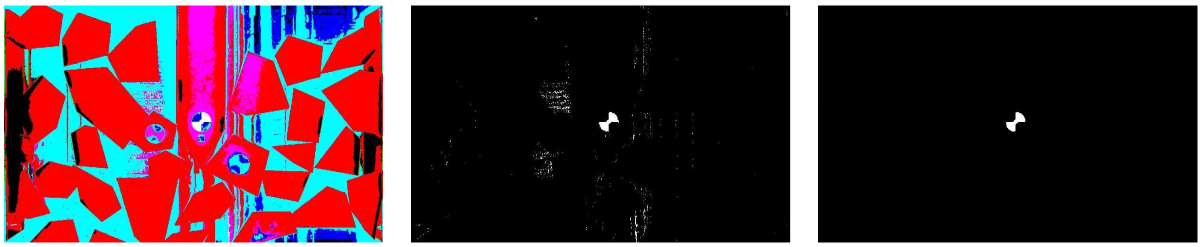
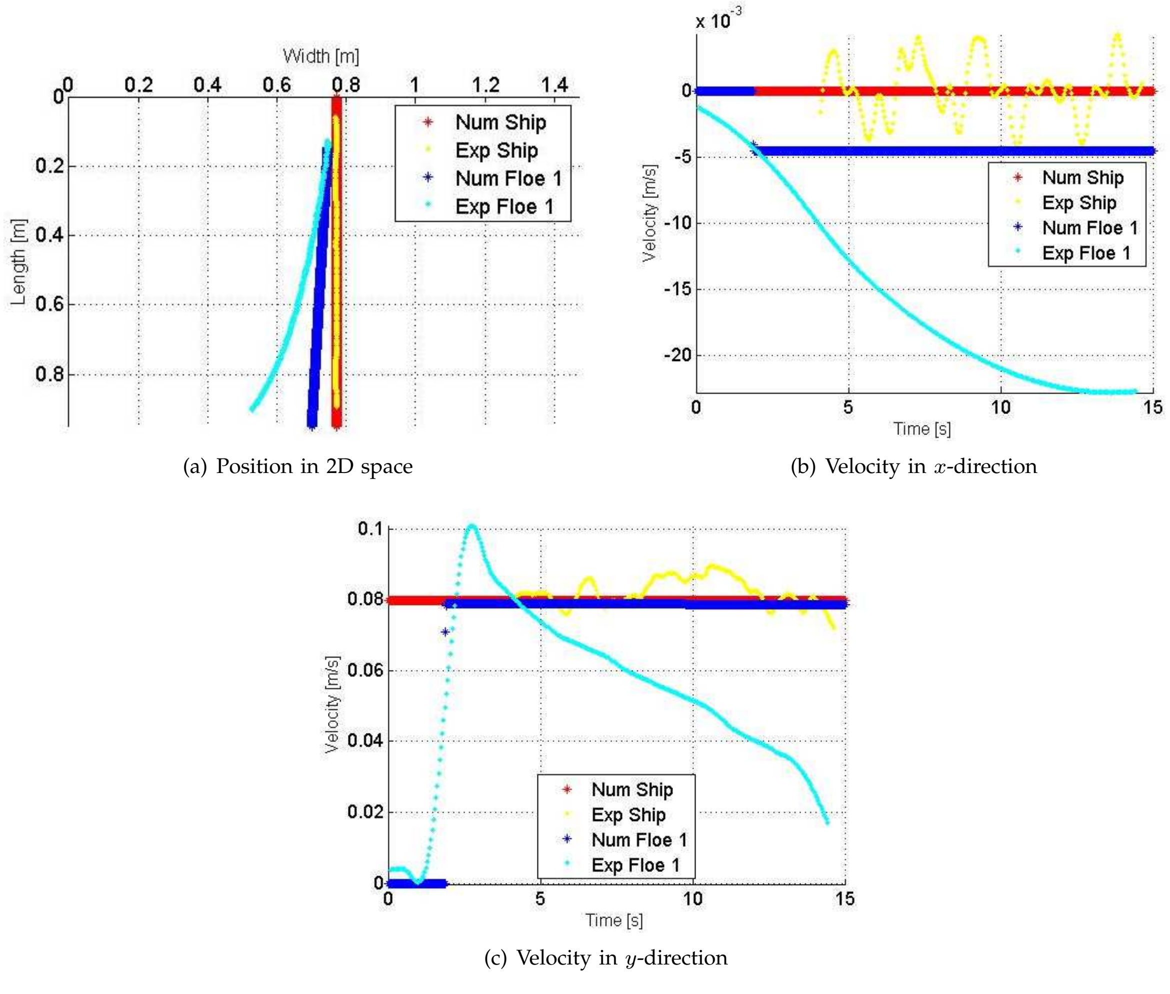

(a) RGB Thresholding (b) Binary Image with noise (c) Final Image

Fig. 17. Three stages of thresholding. First selecting areas based on rgb-colour, converting them to a binary image and removing the noise.



(a) Position in 2D space (b) Velocity in x-direction

(c) Velocity in y-direction

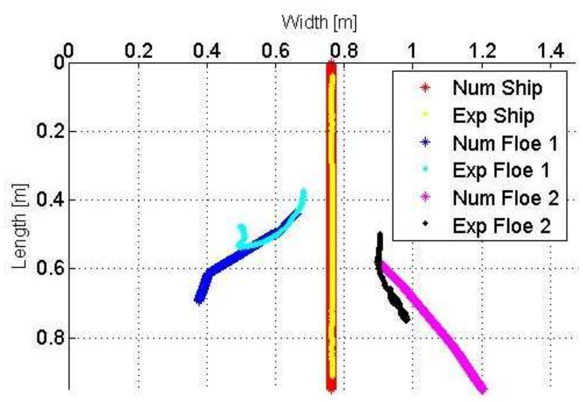
Fig. 18. Comparison between the numerical model (Num) and experimental data (Exp) of an one ship and one floe situation.

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 the open water behaviour.

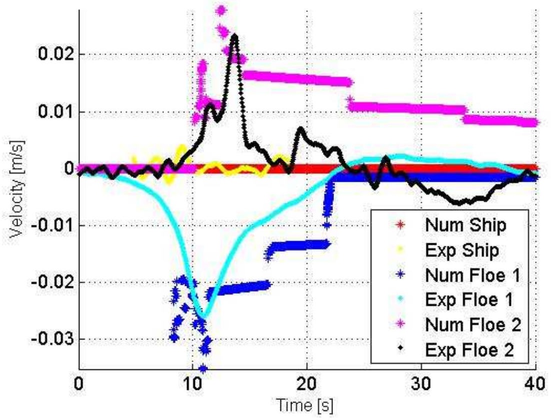
3) In the experiment, it is noticeable that the surface tension makes floes stick together, influencing their motions and speeds. It is clearly seen how the floes follow a different trajectory in Figure 19(a) and 20. This is not incorporated in the model (because in large scale, it is neglectable) but is important in the scale used for the experiments.

7 Conclusion

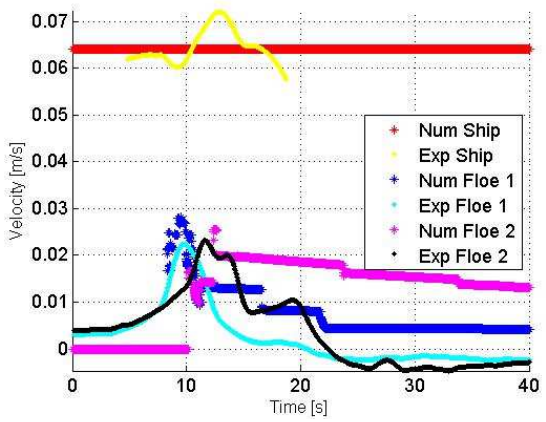
The experiment demonstrated performance benefits for simulating the complex mechanics of a ship operating in pack ice. It is clear that GPGPU has the



(a) Position in 2D space



(b) Velocity in x-direction



(c) Velocity in y-direction

Fig. 19. Pack ice comparison, numerical model (Num) and experimental data (Exp).

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 recent developments in GPU computation 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 GPU 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 experimental data. Especially in the pack ice scenario, it shows realistic behaviour. However, there are some points where the model needs improvement, but the data collected in this research can prove useful when improving the model. First of all, the open water behaviour of the numerical model is not accurately predicted, resulting in an unrealistically high open water velocity of the floe. Secondly, due to the (small) scale of the experiment, surface tension is an important parameter in the floe behaviour, while it is not incorporated in the model. The collisions, however, tend to be modelled more realistically and follow the general trend seen in the experiments.

8 Future Work

While the results so far are promising, we have not yet to reach the point where the simulation is fast enough to be practically used for planning ice management activities in realistic size ice fields. Further development and optimization are necessary to achieve this. One way to achieve a fast enough simulation for a large ice fields is to implement the simulator using multiple GPUs.

The numerical 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 behavior (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.

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.

ACKNOWLEDGMENTS

This research has been done under STePS^2 project and was supported by: ABS, Atlantic Canada Opportunities Agency, BMT Fleet Technology, Husky Oil Operations Ltd, Research and Development Council, Newfoundland and Labrador and Samsung Heavy Industries.

REFERENCES

[1] 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.

[2] Stefan 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*, vol. 12, no. 7-8, pp. 785–826, 2008.

[3] David Tskhakaya, "The particle-in-cell method," in *Computational Many-Particle Physics*, H. Fehske, R. Schneider, and A. Weie, Eds., vol. 739 of *Lecture Notes in Physics*, pp. 161–189. Springer Berlin Heidelberg, 2008.

[4] Haxon, "Ice floe at oslofjord," mar 2009, http://www.panoramio.com/photo/19618780.

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

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

[7] Shadi Alawneh and Dennis Peters, "Ice simulation using gpgpu," in *Proceedings of the 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*, Washington, DC, USA, 2012, HPCC '12, pp. 425–431, IEEE Computer Society.

[8] Shadi Alawneh, Dennis Peters, and Roelof Dragt, "Ice simulation using gpgpu," in *The International GPU Technology Conference (GTC 2013)*. San Jose, California, Poster.

[9] 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 *In Proceedings of Newfoundland Electrical and Computer Engineering Conference (NECEC 2012), IEEE*, St. John's, NL, Canada, Nov 2012.

[10] Claude Daley, Shadi Alawneh, Dennis Peters, Bruce Quinton, and Bruce Colbourne, "Gpu modeling of ship operations in pack ice", international conference and exhibition on performance of ships and structures in ice," in *ICETECH 2012*, Banff, Alberta, Canada, 2012.

[11] Claude Daley, Shadi Alawneh, Dennis Peters, and Bruce Colbourne, "Gpu-event-mechanics evaluation of ice impact load statistics," in *Submitted To Offshore Technology Conference (OTC 2014)*, Houston, Texas, USA, 2014.

[12] E. Enkvist, P. Varsta, and K. Riska, "The ship-ice interaction," in *In Proceedings of The International Conference on Port and Ocean Engineering under Arctic Conditions (POAC*.

[13] T.V. Kotras, A.V. Baird, and J.N. Naegle, "Predicting ship performance in level ice," *SNAME Transactions 91*, p. 329349, 1983.

[14] P. Valanto, "The resistance of ships in level ice," *SNAME Transactions 109*, p. 5383, 2001.

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

[16] A.J. Keinonen, Browne R.P. Revill, and A. Reynolds, "Ice breaker characteristics synthesis," Tech. Rep. TP 12812 E., Report of AKAC Inc. to Transportation Development Centre, 1996.

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

[18] Raed Lubbad and Sveinung Løset, "A numerical model for real-time simulation of shipice interaction," *Cold Regions Science and Technology*, vol. 65, no. 2, pp. 111 – 127, 2011.

[19] Peter Kipfer, Mark Segal, and Rüdiger Westermann, "Uber-Flow: a GPU-based particle engine," in *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, New York, NY, USA, 2004, pp. 115–122, ACM.

[20] 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*, New York, NY, USA, 2004, HWWS '04, pp. 123–131, ACM.

[21] Simon Green, "Nvidia particle system sample," 2004, http://download.developer.nvidia.com/developer/SDK/.

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

[23] C.G. Daley, "Energy based ice collision forces," in *POAC '99*, Helsinki, Finland, 1999.

[24] C.G. 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.

[25] Nvidia, "Cuda development tools v2.3. getting started," 2009.

[26] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron, "Scalable parallel programming with cuda," *Queue*, vol. 6, pp. 40–53, March 2008.

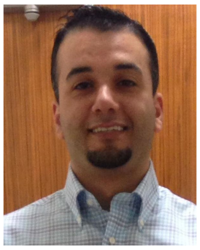
[27] David Eberly, "Intersection of convex objects: The method of separating axes," *Geometric Tools, LL*, 2008.

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

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

[30] Christer Ericson, *Real-Time Collision Detection*, Morgan Kaufmann, 2005.

[31] Nadathur Satish, Mark Harris, and Michael Garland, "Designing efficient sorting algorithms for manycore gpus," in *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, Washington, DC, USA, 2009, IPDPS '09, pp. 1–10, IEEE Computer Society.



Shadi Alawneh received the B. Eng. degree in computer engineering from Jordan University of Science and Technology, Irbid, Jordan in 2008, the M. Eng. and PhD degrees in computer engineering from Memorial University of Newfoundland, St. John's, NL, Canada in 2010 and 2014, respectively. Then, he joined the Hardware Acceleration Lab at IBM Canada as a Staff Software Developer from May 2014 through August 2014. He is currently a research engineer at Smart Solutions for Challenging Environments (C-CORE), St. John's, NL, Canada. His research interests include parallel and distributed computing, general purpose GPU computing, numerical simulation and modeling, software optimization and computational geometry. He is a young professional member of the IEEE.

Roelof Dragt received the BSc. degree in marine engineering from Delft University of Technology, Delft, The Netherlands in 2011 and the MSc. Degree in offshore engineering from Delft University of Technology in 2013. Since 2013, he is working as Scientist at TNO (Netherlands Organization for Applied Scientific Research) within the Structural Dynamics department. He conducts both experimental as theoretical research, specializing in failure mechanisms in the offshore and maritime environment.

Dennis Peters received the B.Eng. (Electrical) degree at Memorial University from 1984 through 1990. Following this he moved to Ottawa to work in the high-tech industry with Newbridge Networks (now Alcatel). After two years in industry he returned to school, this time at McMaster University in Hamilton, Ontario where he completed the M.Eng. (Electrical & Computer) in 1995 and PhD (Electrical & Computer Engineering) in 2000. He joined the faculty at Memorial in 1998. His teaching is primarily in the area of software, ranging from introductory programming courses to advanced topics such as software engineering and concurrent programming. His research is concerned with developing techniques and tools to facilitate the production, analysis and use of documentation for computer system behavioural requirements and design. Such documentation must be both 1) clear enough to be read and understood, with a minimum of special training, by both domain experts and programmers, and 2) complete and precise enough to allow thorough analysis, by manual or automatic means. He is a senior member of the IEEE.

Claude Daley received the B.Eng. degree from the University of Western Ontario in 1977 (civil engineering), then he went on to Princeton University to complete a master's degree in structures and mechanics. In 1989, he was posted to Helsinki in a collaborative research project between Canada and Finland. During three years in Helsinki, he obtained a doctorate of technology in the area of ice mechanics and arctic naval architecture. Then, in 1995 he joined Memorial University. His current research focuses on ice mechanics and ice loads on ships; plastic strength and design of ship structures; concepts for rational ship structural design regulations; structural risk; and related matters. He is a member of the ISSC (International Ship Structures Committee - member of the committee on Condition Assessment of Aged Ships). He is also currently on the Board of Examiners of the Professional Engineers and Geoscientists of Newfoundland and Labrador, on a curriculum committee of the Canadian Council of Professional Engineers and he serve on the Executive Committee of the Senate of Memorial University.

Steve Bruneau a graduate of Memorial University's civil engineering program in 1987, he worked for a few years in the structural steel and construction business before heading to the University of Western Ontario, Boundary Layer Wind Tunnel. There, he worked on a few wind related studies most notable of which are the topsides wind analysis for the Hibernia Production Platform and the pedestrian level wind analysis for the Sears Tower in Chicago. Further studies in fluids led to an M.E.Sc in Industrial Aeronautics and Hydrodynamics. In 1992, he returned to Newfoundland to work at C-CORE, immediately becoming immersed in the ongoing iceberg design load work for the Terra Nova floating production system. This work prompted his PhD studies at Memorial University in ice loads, but not long after startup, ice issues at the Confederation Bridge took center stage and that ended up being the final theme of his doctoral thesis in 1996. Subsequent work at C-CORE involved de-icing systems for microwave antennas and various iceberg engineering and management undertakings. He joined the Faculty of Engineering and Applied Science in January 2006 as an assistant professor in civil engineering and he intend to focus his R&D work in niche areas of ice, wind, hydro and energy disciplines as they relate to development in Newfoundland and Labrador.