Modeling, Identification and Control, Vol. 35, No. 4, 2014, pp. 211–248, ISSN 1890–1328

A Software Framework for Simulating Stationkeeping of a Vessel in Discontinuous Ice

Ivan Metrikin

Department of Civil and Transport Engineering, NTNU, N-7491 Trondheim, Norway

Arctic Design and Operations, Statoil ASA, N-7053 Trondheim, Norway

Abstract

This paper describes a numerical package for simulating stationkeeping operations of an offshore vessel in floating sea ice. The software has found broad usage in both academic and industrial projects related to design and operations of floating structures in the Arctic. Interactions with both intact and broken ice conditions can be simulated by the numerical tool, but the main emphasis is placed on modelling managed ice environments relevant for prospective petroleum industry operations in the Arctic. The paper gives a thorough description of the numerical tool from both theoretical and software implementation perspectives. Structural meshing, ice field generation, multibody modelling and ice breaking aspects of the model are presented and discussed. Finally, the main assumptions and limitations of the computational techniques are elucidated and further work directions are suggested.

Keywords: global ice loads, numerical simulation, physics engine, computational geometry, packing algorithm

Nomenclature

α	Orientation angle of a polygon
α_{wedge}	Ice wedge opening angle
β	Error reduction parameter
λ	Geometrical scaling factor
$\{I_{polygon}\}$	List of intersection polygons between the vessel and an ice floe polygon
$\{S_{polygon}\}$	List of intersection polygons between the vessel and the top plane of an ice floe
μ_k	Coefficient of kinetic friction
μ_s	Coefficient of static friction
ρ	Density of a medium
σ_c	Compressive strength of ice
σ_f	Flexural strength of ice
σ_{wedge}^{max}	Maximal bending stress in a wedge-shaped ice beam resting on an elastic foundation
σ_{yy}	Bending stress in a semi-infinite ice sheet
θ_{crack}	Polar angle of the splitting crack
$\triangle t$	Time step
$\vec{\lambda}$	Vector of Lagrange multipliers of the multibody system
$\vec{\Omega}$	Angular velocity of a body
$\vec{\tau}$	Torque acting on a body
$\vec{b}_{box}$	Object-aligned bounding box vector
$\vec{b}_{pen}$	Position correction vector of the multibody system

$\vec{d}_{best}$	Maximal scaling factor for a polygon in the ice field generation algorithm
$\vec{d}_{max}$	Maximal floe size
$\vec{d}_{min}$	Minimal floe size
$\vec{d}_{scale}$	Scaling vector in the ice field generation algorithm
$\vec{F}$	Force vector of the multibody system
$\vec{f}$	Force acting on a body
$\vec{g}$	Acceleration of gravity
$\vec{n}$	Contact normal
$\vec{n}_{\triangle}$	Normal to an intersection point between the vessel and an ice floe
$\vec{n}_{break}$	First possible crack propagation direction
$\vec{n}_{polygon}$	Normal of an intersection polygon between the vessel and an ice floe
$\vec{n}_{split}$	Splitting crack direction vector
$\vec{p}$	Constraint impulse vector
$\vec{r}$	Coordinate of a point
$\vec{u}$	Velocity vector of the multibody system
$\vec{v}$	Linear velocity of a body
A	Area of a geometrical shape
b	Thickness of an artificial boundary around the ice field region
C_a	Angular drag coefficient
C_d	Form drag coefficient
C_s	Skin friction coefficient
d_{lim}	Ice floe narrowness threshold
d_{pen}	Penetration depth in a contact point
e	Coefficient of restitution
F	Friction coefficient matrix of the multibody system
H	Kinematic map matrix of the multibody system
h_i	Ice thickness
I	Inertia tensor of a body
$I_{3\times3}$	3-by-3 unit matrix
J	Constraint Jacobian matrix of the multibody system
j_{pos}	Positional noise in the ice field generation algorithm
j_{rot}	Rotational noise in the ice field generation algorithm
k_d	Constraint damping factor
k_s	Constraint stiffness factor
L	Length of a region
l	Length of an object
l_{avg}	Average length of a fluid mesh
$L_{breakable}$	Ice floe breakability length
M	Inertia matrix of the multibody system
m	Mass of an object
N_{body}	Amount of bodies in the simulation
$N_{contact}^{inters}$	Amount of contact points associated with an intersection polygon between the vessel and an ice floe
N_{cont}	Amount of contacts in the multibody system
N_{expand}	Amount of attempts to expand a polygon in the ice field generation algorithm
N_{fails}	Amount of attempts to place a polygon in the ice field generation algorithm
N_{joint}	Amount of joints in the multibody system
N_{size}	Amount of the floe size sub-intervals in the ice field generation algorithm
N_{trials}	Amount of attempts to perturb a polygon in the ice field generation algorithm
N_{ver}	Amount of polygon vertices in the ice field generation algorithm
N_{wedges}	Amount of ice wedges in a circumferential crack
p_{lim}	Limiting impulse of the ice crushing constraint
Q	A quaternion
q_{load}	Areal density of the distributed load acting on a semi-infinite ice plate
R	Rotation matrix of a body
r, p, q	Cardan angles

$R_{bending}$	Circumferential crack radius
R_{load}	Radius of the loaded area on a semi-infinite ice plate
S_k	Splitting radius multiplication coefficient
T	Transformation matrix of a body
t	Length of a line segment in the polygon generation algorithm
V	Volume of a geometrical shape
v_{proj}	Linear velocity projection on a surface
W	Width of a region
w	Width of an object
X, Y, Z	Axes of a Cartesian reference frame
x, y, z	Coordinates in a Cartesian reference frame

1 Introduction

The majority of prospective hydrocarbon deposits in the Arctic are located offshore, beyond the 100-m water depth contour, which is considered to be the upper limit for bottom-founded structures in areas with possible sea ice intrusions (Hamilton, 2011). Operations in such areas require robust floating drilling and production systems, supported by a reliable fleet of ice management, supply, emergency response and intervention vessels. Upcoming oil and gas activities require these floating structures to withstand environmental loads, including those from sea ice.

Although it is anticipated that the first upcoming oil and gas operations in deep-water Arctic areas will take place primarily within the open water season, sea ice intrusions may occur under some circumstances. If this happens, it is expected that the ice management fleet will be able to break down the incoming ice floes into acceptable sizes, such that the stationkeeping performance of the platform would not be compromised. Therefore, the ice cover approaching the operational site will be *discontinuous*, i.e. broken into discrete ice features of various shapes and sizes.

The ability of a platform to hold its position during a sea ice intrusion event is governed by the relationship between the global environmental load from broken ice and the stationkeeping system capacity. The ice load depends on the structure-ice interaction process which involves complex contact mechanics: ice material failure, rigid-body motion of the broken ice pieces, ice-ice and ice-structure friction and fluid effects. Moreover, the boundary conditions of the managed ice domain have an influence on the load-response relationship of the dynamical system, leading to highly nonlinear and complex behavior.

Full-scale data obtained during the Kulluk platform operations (Wright, 1999) indicates that stationkeeping in managed ice is possible, and the mooring system loads measured during Kulluk operations can be used to perform analytical estimates of the global ice loads for design purposes (Palmer and Croasdale, 2013). However, such estimates have not been validated by full-scale data for the case of ship-shaped structures, which are currently considered attractive for Arctic deep-water exploration. Therefore, ice tank experiments are usually performed on new designs to establish the global loads and operational envelopes of floating offshore structures in various sea ice conditions.

Although model testing is currently considered to be the state-of-the-art method for estimating global ice loads, it has some limitations. Scaling uncertainties and boundary effects are the most challenging ones when testing floaters in broken ice, especially vertically-sided ship-shaped structures. For example, the author of this paper has experienced a 7-meter-long vessel moored at 30° oblique angle relative to the ice drift to induce a boundary interaction in a 10-meter-wide ice tank after just first 3 meters of a model experiment. Therefore, it seems attractive to develop a numerical model for simulating such interactions, validate it against model-scale data, and then use the validated model to expand the testing environment (for example to increase the width of the ice basin). Moreover, such model could be used for pre-simulating the model tests and optimizing the amount of physical experiments in advance of the actual testing campaign.

Such numerical model has been developed within the framework of the DYPIC (Kerkeni et al., 2014) and Arctic Dynamic Positioning (DP) (Skjetne, 2014) projects. The initial goal of the model was to run more scenarios than during the laboratory testing campaign of the DYPIC project at the large ice tank of the Hamburg Ship Model Basin (HSVA). Therefore, the model was given a preliminary name "Numerical Ice Tank", as it was tailored to replicate model-scale experiments at HSVA. However, the latest version of the model supports also full-scale simulations, as reported in Scibilia et al. (2014).

To the best of the author's knowledge, the first numerical ice tank concept was proposed by Valanto and Puntigliano (1997) to simulate the icebreaking resistance of a ship in level ice. That paper suggested an approach which separated the icebreaking process at the design waterline of a vessel from the motions of the broken ice floes under the hull. Later, Derradji-Aouat (2010) described a fully coupled numerical ice tank environment based on Explicit Finite Element Method implemented within commercial packages ANSYS and LS-DYNA. This technique was used by Wang and Derradji-Aouat (2010) and Wang and Derradji-Aouat (2011) to simulate model-scale and full-scale interactions of structures with broken ice. However, although the approach proposed by Derradji-Aouat (2010) is generic, the ice pieces were considered unbreakable in those examples. Finally, Lee et al. (2013) proposed a numerical ice tank based on the multi-material arbitrary Lagrangian formulation and the fluid-structure interaction analysis technique of the LS-DYNA code. Pre-sawn level ice tests of 2 different hull shapes were successfully simulated in that paper.

The main differences of the current model from the other numerical ice tanks is the assumption of low interaction velocities between the vessel and the ice (i.e. below 2 m/s in full-scale), and the focus on modelling broken ice conditions. The former assumption leads to a simplified treatment of hydrodynamic interactions, because the ice breaking phenomena become more a problem of solid mechanics rather than hydrodynamics. The latter leads to adoption of the physics engine software for collision detection, contact force computation and time stepping of the numerical model. From the theoretical perspective, the physics engine approach is equivalent to the nonsmooth discrete element method (Metrikin and Løset, 2013). The main difference of the nonsmooth discrete element method from the conventional penalty-based discrete element method is that the stiffness and damping parameters do not need to be introduced into the contact problem formulations. Therefore, larger time steps can be used and a higher computational efficiency can be achieved in the numerical tool. A detailed comparison of the two different discrete element methods can be found in e.g. Servin et al. (2014). Discrete treatment of the ice features allows the model to calculate the ice loads due to breaking, rotation, submergence and sliding explicitly, i.e. the motions of the broken ice floes under and around the hull are fully modelled. This is another major difference of the current model from the commonly used ice material transport formulations of Lindqvist and Croasdale (Su, 2011; Bonnemaire et al., 2011). The simulated ice failure modes in the current model include crushing, flexural bending and splitting. Therefore, although the model is tailored to simulating broken ice conditions, it is also capable of modelling fixed and floating offshore structures interacting with intact ice.

The software package has been broadly used by the industry and academia to investigate design and operations of floating Arctic offshore structures. Metrikin et al. (2013b) used the first version of the model for simulating dynamic positioning of an Arctic drillship in managed ice and compared the modelling results with model testing data. Later, Kerkeni et al. (2013a) used the model to compare the DP control laws in open water and ice, and to establish DP capability plots of a vessel in managed ice (Kerkeni et al., 2013b). Metrikin and Løset (2013) simulated an oblique towing test of an Arctic drillship in managed ice and compared simulation results to experimental data. Later, Metrikin et al. (2013a) performed the first simulation of DP in level ice and Kerkeni and Metrikin (2013) proposed an automatic heading controller for managed ice conditions, together with its verification by numerical simulations. Finally, Scibilia et al. (2014) simulated the icebreaker Oden in full-scale broken ice conditions offshore North-East Greenland, Østhus (2014) developed an adaptive control methodology for a vessel in managed ice using hybrid position and force control, and Kjerstad and Skjetne (2014) performed modeling and control of a DP vessel in curvilinearly drifting managed ice. Full development timeline of the model is outlined in Kerkeni et al. (2014).

Currently, the binary version of the software has been released to commercial partners of the DYPIC and Arctic DP projects, while the source code and intellectual property rights of the model are owned by the Norwegian University of Science and Technology.

This paper describes the numerical tool from both theoretical and software implementation perspectives. First, Section 2 gives a high-level overview of the simulation model and its main components. Next, Section 3 describes the preparation steps the user has to take for executing a simulation run. Then, Section 4 outlines the initialization sequence of the numerical model, including structural creation and ice field generation. Afterwards, Section 5 specifies the theoretical aspects of the actual simulation process, including the multibody dynamics model, the fluid force model and the ice breaking model. Then, Section 6 presents the output functionalities of the simulator. Finally, Section 7 discusses the main assumptions and limitations of the computational approach, and Section 8 summarizes and concludes the paper.

2 Simulator Overview

On the highest level the simulator is structured as shown in Figure 1. The user sets up a simulation in the software by creating an input file for the pre-processor. The structure and contents of this file are discussed in Section 3.1. Then, the user should perform the meshing of the simulated vessel according to a special process outlined in Section 3.2. Finally, the actual simulation program starts by initializing the physical scene (Section 4) and entering the simulation loop (Section 5).

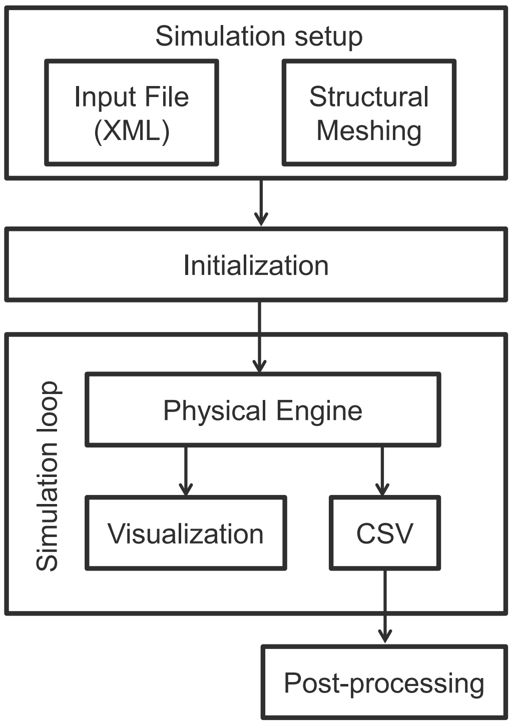

Figure 1: Simulation workflow of the numerical model.

The simulation loop is centered around the physical engine middleware. This software development kit (SDK) is used to perform collision detection, contact manifold creation, rigid body and joint dynamics computation, contact force evaluation and time integration of the equations of motion. Metrikin et al. (2012b) described a framework for modelling a floating structure in broken ice using a physical engine, and Metrikin et al. (2012a) performed a comparative study of different physical engines for simulating ice-structure interactions. Later it was realized that physical engines are being constantly developed and improved, and one solution might become outdated in a very short time. Therefore, it was decided to base the software package on a generic interface that would support any physical engine, so that the user could easily switch between the different engines and integrate new ones when necessary. Such interface has been implemented using the Physics Abstraction Layer (PAL) library (Boeing, 2009), which provides a unified interface to many different physical engines. The PAL library is implemented as a C++ abstract pluggable factory which maintains a common registry of physical objects that can be simulated by all underlying physical engines. When a certain engine is selected by the user, the registry is populated with objects and methods for that specific engine during run-time of the application. Further description of the physical engine itself, including the theoretical background, is given in Section 5.

Optional visualization of the simulation scene can be performed by the software in real-time using the Irrlicht library (Gebhardt et al., 2012) (Section 6.2). Furthermore, the simulator can produce numerical output data in comma-separated format (CSV) for subsequent analysis and post-processing by the user (Section 6.1).

The full numerical model is packaged into a 32-bit double-precision Microsoft Windows application. It has been developed in C++ programming language using Microsoft Visual Studio 2010 development environment, and the Microsoft Visual C++ runtime is needed to run the software on the user's machine. The C++ Standard Library has been used for memory management (e.g. smart pointers), exception handling, input/output streaming, textual string processing, numerical functions (e.g. square root), container management (e.g. vectors, maps, sets and their algorithms) and templates. Version 3 of the Boost.Filesystem library (Dawes, 2012) has been used to manipulate files, directories and system paths for the majority of input/output tasks in the software. Finally, the build system of the application has been implemented using Windows batch scripting, and the source code lifecycle has been managed using the Subversion (SVN) technology.

3 Simulation Setup

This section describes the pre-processing steps required for the user to set up a simulation in the software package.

3.1 Input File Preparation

The input file for the software tool has an Extensible Markup Language (XML) format and contains all essential information for initializing the numerical model. Loading and parsing of the input file is implemented using version 1.2 of the PugiXML library (Kapoulkine, 2014). There are 3 sections in the input file: overall simulation settings, scene settings and output settings. The contents of these sections are described in the following paragraphs.

Firstly, the overall simulation settings contain the name of the physical engine to use in the simulation loop. The user can select between AgX Dynamics (Algoryx Simulation AB, 2014), Bullet Physics (Coumans, 2012), Open Dynamics Engine (Smith, 2014), NVIDIA PhysX (Nvidia Corporation, 2014) and Vortex Dynamics (CM Labs Simulations Inc., 2014) libraries. However, the ice breaking functionality is currently supported only in the Bullet-based implementation of the numerical tool. Therefore, if the user selects any engine except Bullet, the ice floes will be treated as unbreakable (see Section 5 for more details on ice breaking).

Furthermore, the overall simulation settings section of the input file contains the amount of CPU threads to run (the application supports multithreading), the size of the time step, the total amount of time steps to simulate and the real-time visualization system settings (i.e. the target, heading, elevation and distance to the camera in the global spherical coordinate system).

The scene settings of the input XML file contain the following parameters:

- Properties of the simulated vessel: geometry, mass, inertia tensor and initial position/orientation;
- Physical size of the virtual ice tank: length, width and depth;
- Acceleration of gravity defined as a 3D vector in the global reference frame;
- Ice properties: density, thickness, compressive and flexural strength values and Youngs modulus;
- Fluid properties: density, drag coefficients and type of the angular drag model (see Section 5.3);
- Mechanical contact properties for the various pairs of contact surfaces: ice-ice, ice-structure and ice-boundary (e.g. static and dynamic friction coefficients, possible restitution and stiffness/damping in constraints - see Section 5);
- The ice field is defined inside a specified rectangular region which is characterized by the type of the ice feature (intact or broken), target ice concentration in %, floe size distribution and the seed of random number generator for the packing algorithm (Section 4.3);
- Simulation type: free running, oblique towing or dynamic positioning (DP). The DP system interface is implemented as a static library component (a set of .lib files) which can be linked by the user into a custom-made control system project. Both the simulation and visualization processes can be controlled by the user through this interface. At the start of every time step of a DP simulation the simulator receives actuation inputs in surge, sway and yaw DOFs and sums them additively with any other external forces acting on the vessel to obtain the total load (see Metrikin et al. (2013b), Kjerstad and Skjetne (2014) and Section 5). A mooring system or any other external forces acting on the vessel can also be implemented through this DP interface.

Finally, in the output settings section of the input XML file the user specifies the system path to the output CSV file; the components of the position, orientation, force and torque vectors of the vessel to record at every time step (computed in the global reference frame, as described in Section 6.1); and the amount of numerical digits after the decimal point for all output values (the software can output numerical values up to machine precision).

3.2 Structural Meshing

The geometry of the vessel is also specified in the scene settings section of the input XML file. In the simulation it is represented by 2 different triangulated surface meshes: a fluid mesh (Figure 2) and a collision mesh (Figure 3). If the user would like to run model-scale simulations, it is possible to specify a scaling factor for the meshes in the input XML file. The simulator will then grow or shrink both meshes to the desired geometrical scaling factor λ.

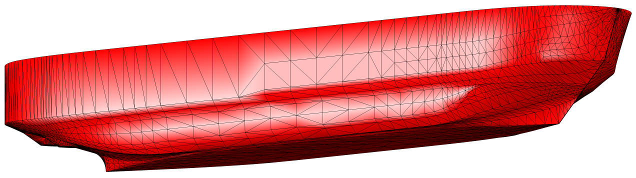

Figure 2: Fluid mesh of a conceptual Arctic drillship (courtesy Statoil).

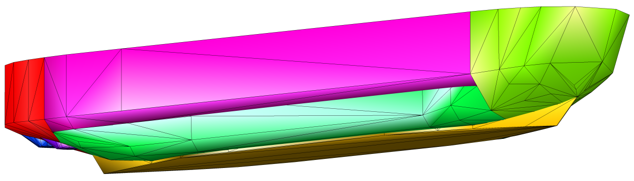

Figure 3: Collision mesh of a conceptual Arctic drillship (courtesy Statoil). Different colors represent different convex decomposition pieces.

The fluid mesh is used for calculating the inertia tensor of the vessel and the fluid forces acting on it (see Section 5.3), while the collision mesh is used to construct a geometrical representation of the vessel for detecting contacts with the ice floes in the physical engine. The simulator does not support concave collision meshes, so if there are any concavities in the original collision mesh, it must be decomposed into convex elements (otherwise the collision detection system of the physical engines will either fail or work very slowly).

Some automatic convex decomposition tools are available for this task, e.g. the Hierarchical Approximate Convex Decomposition tool (Mamou, 2013). However, experience shows that although such tools can provide a reasonable initial convex decomposition, the final refinement has to be performed manually using a 3D computer graphics software package (such as 3ds Max (Autodesk, 2014) or Blender (Blender Foundation, 2014)). Throughout the manual refinement process it is very important to constantly ensure that when the convex pieces are added together they will represent the original mesh as closely as possible. Common practice is to use the original concave fluid mesh as a reference for the convex-decomposed collision mesh.

The geometrical file format used by the simulator is Wavefront OBJ, and files constituting both the fluid and collision meshes should be supplied in this format. The physical engines need only the vertex coordinates of the convex collision meshes for their construction, while fluid meshes require the face indices in addition to vertex coordinates. Currently, due to limitations of the OBJ file loader in the application, all meshes can have only triangular faces (the file loader is implemented using an open source library developed by Lindenhof (2012)). Furthermore, before starting a simulation it is recommended to pre-process the meshes so that they contain as few faces as possible, in order to reduce the computational burden on the simulator. Moreover, the user must ensure that the meshes are closed (i.e. every edge has exactly 2 adjacent triangular faces) and do not contain any holes, otherwise the collision detection system will have artifacts. Finally, degenerate features, such as too narrow triangles or duplicate surfaces, must be removed from the meshes in order to avoid degenerate collision configurations. It is worth mentioning that the author's experience in using the simulator indicates that mesh preparation is currently the most time-consuming part of the whole simulation setup process.

4 Initialization of the Simulation Scene

The simulation scene in the numerical model contains 5 physical components: the fluid domain, the vessel, the ice field, the boundaries and, optionally, the towing carriage (Figure 4). The global coordinate system $\{X_g, Y_g, Z_g\}$ is assumed inertial, while local coordinate systems of the structure $\{X_s, Y_s, Z_s\}$ and the ice floes $\{X_i, Y_i, Z_i\}$ are moving attached to their respective objects.

Initialization of the scene is performed stepwise, starting from creation of the boundaries which demarcate a rectangular region in the $\{X_g, Y_g\}$ plane where all simulated objects will be confined (Figure 5). The length L and width W of this region are specified by the user in the input file. The boundaries are represented by static rectangular cuboids in the simulation, meaning that they constitute immovable rigid objects in the physical engine. The height of the cuboids defines the depth of the virtual ice tank, as specified by the user in the input file.

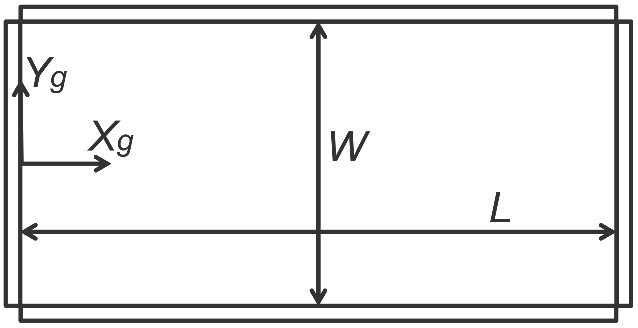

Figure 5: Boundaries of the simulation domain.

Next, the fluid domain is created. It is represented by a static half-space demarcated by the $\{X_g, Y_g\}$ plane, i.e. the water level is always at $Z_g = 0$. The whole underwater domain is assigned a uniform constant density, as specified by the user in the input file. All objects in contact with the fluid are affected by buoyancy and drag forces which are computed according to a method described in Section 5.3.

The following sub-sections describe the creation of the vessel (Section 4.1), the towing carriage (Section 4.2) and the ice field (Section 4.3). However, before these objects are constructed, the simulator initializes the contact property table in the selected physical engine using the PAL interface. For every couple of contacting surfaces (ice-ice, ice-structure and ice-boundary), the following values are populated from the input file: static friction coefficient μ_s, kinetic friction coefficient μ_k, restitution coefficient e, constraint stiffness factor k_s and constraint damping factor k_d. When 2 objects experience a pairwise collision this table is used to define their interaction mechanics according to the contact dynamics method (Section 5).

4.1 Vessel Creation

Creation of the vessel in the numerical model is a stepwise process. First of all, the fluid and collision meshes are loaded from their respective Wavefront OBJ files, and geometrical scaling of the vertices is performed: $\vec{r}_i^{\,new} = \lambda \vec{r}_i^{\,old}$, where $\vec{r}_i^{\,old}$ is the original 3D coordinate of vertex i in the local coordinate system of the mesh,

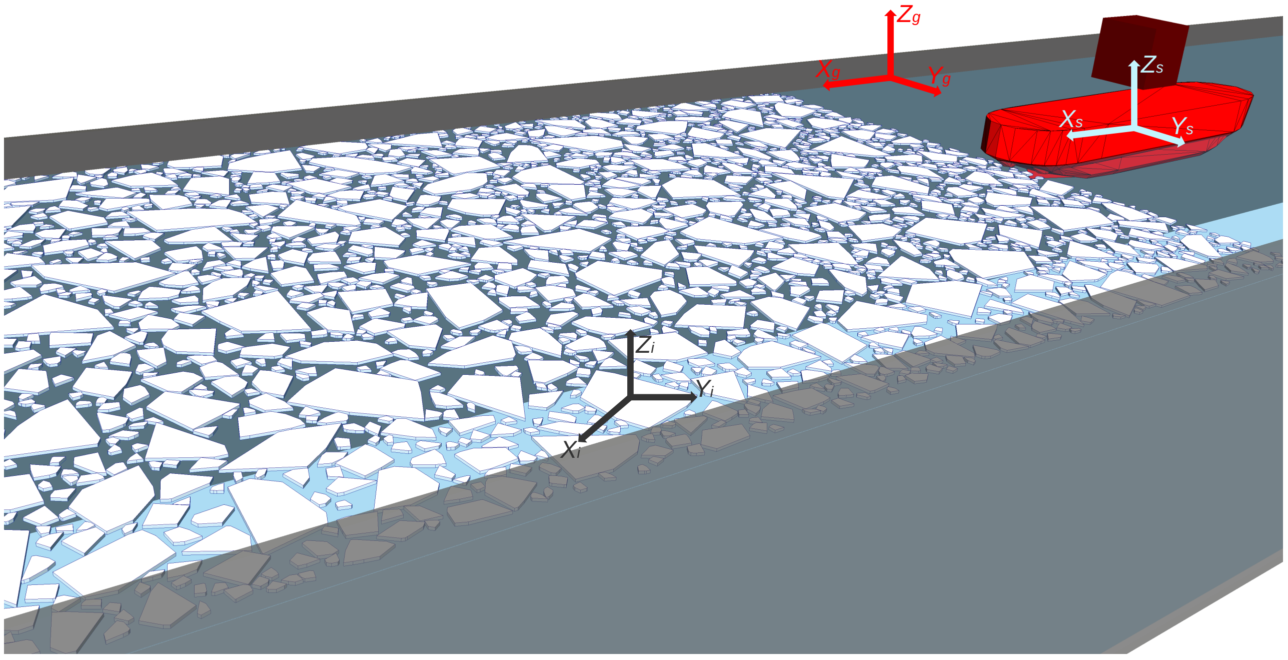

Figure 4: Simulation scene of the numerical model.

λ is the geometrical scaling factor from the input file and $\vec{r}_i^{new}$ is the scaled coordinate of the vertex. After the scaling operation, each convex decomposition piece is converted into a point cloud with unique vertices, because the convex mesh loader of the application removes triangulation and discards duplicate vertices from the collision meshes of the vessel.

Secondly, the body of the vessel is created in the physical engine through the PAL interface. Its collision geometry becomes a compound shape consisting of the provided convex pieces, and the fluid mesh is used to calculate the inertial properties of the body according to the following algorithm:

1. The signed volume of each tetrahedron in the fluid mesh is computed as follows:

$$V_{\triangle}^i = \frac{1}{6}\vec{r}_1^{\,i} \cdot [(\vec{r}_2^{\,i} - \vec{r}_1^{\,i}) \times (\vec{r}_3^{\,i} - \vec{r}_1^{\,i})]$$

   where $\vec{r}_{1,2,3}^{\,i}$ are 3D vertex coordinates of triangle i in the local coordinate system of the mesh, and the apex of the tetrahedron is assumed to coincide with the origin of this coordinate system. The $\cdot$ sign represents vector dot product, and the $\times$ sign represents vector cross product.

2. Center of mass of every tetrahedron is calculated in the local coordinate system of the mesh as follows:

$$\vec{r}_{CoM}^{\,\triangle_i} = \frac{1}{4}(\vec{r}_1^{\,i} + \vec{r}_2^{\,i} + \vec{r}_3^{\,i})$$

3. The total volume of the mesh is calculated by accumulating the signed volumes of every tetrahedron:

$$V = \sum_i V_{\triangle}^i$$

4. The center of mass of the whole mesh is calculated in the local coordinate system of the mesh as follows:

$$\vec{r}_{CoM} = \frac{1}{V}\sum_i \vec{r}_{CoM}^{\,\triangle_i} V_{\triangle}^i$$

5. An auxiliary 3D vector $\vec{d}$ is computed for every triangle in the mesh:

$$\vec{d}^{\,i} = (\vec{r}_1^{\,i})^2 + (\vec{r}_2^{\,i})^2 + (\vec{r}_3^{\,i})^2 + \vec{r}_1^{\,i} \circ \vec{r}_2^{\,i} + \vec{r}_1^{\,i} \circ \vec{r}_3^{\,i} + \vec{r}_2^{\,i} \circ \vec{r}_3^{\,i}$$

   where the $\circ$ sign represents the Hadamard vector product.

6. Components of the inertia tensor of each unit-mass tetrahedron are computed in the local coordinate

system of the mesh using the method of Tonon (2005):

$$I_{11}^{\triangle_i} = \frac{d_y^i + d_z^i}{10}$$

$$I_{22}^{\triangle_i} = \frac{d_x^i + d_z^i}{10}$$

$$I_{33}^{\triangle_i} = \frac{d_x^i + d_y^i}{10}$$

$$I_{12}^{\triangle_i} = I_{21}^{\triangle_i} = -\frac{1}{20}\cdot$$
$$\cdot(2r_{1,x}^i r_{1,y}^i + r_{2,x}^i r_{1,y}^i + r_{3,x}^i r_{1,y}^i +$$
$$+ r_{1,x}^i r_{2,y}^i + 2r_{2,x}^i r_{2,y}^i + r_{3,x}^i r_{2,y}^i +$$
$$+ r_{1,x}^i r_{3,y}^i + r_{2,x}^i r_{3,y}^i + 2r_{3,x}^i r_{3,y}^i)$$

$$I_{13}^{\triangle_i} = I_{31}^{\triangle_i} = -\frac{1}{20}\cdot$$
$$\cdot(2r_{1,x}^i r_{1,z}^i + r_{2,x}^i r_{1,z}^i + r_{3,x}^i r_{1,z}^i +$$
$$+ r_{1,x}^i r_{2,z}^i + 2r_{2,x}^i r_{2,z}^i + r_{3,x}^i r_{2,z}^i +$$
$$+ r_{1,x}^i r_{3,z}^i + r_{2,x}^i r_{3,z}^i + 2r_{3,x}^i r_{3,z}^i)$$

$$I_{23}^{\triangle_i} = I_{32}^{\triangle_i} = -\frac{1}{20}\cdot$$
$$\cdot(2r_{1,y}^i r_{1,z}^i + r_{2,y}^i r_{1,z}^i + r_{3,y}^i r_{1,z}^i +$$
$$+ r_{1,y}^i r_{2,z}^i + 2r_{2,y}^i r_{2,z}^i + r_{3,y}^i r_{2,z}^i +$$
$$+ r_{1,y}^i r_{3,z}^i + r_{2,y}^i r_{3,z}^i + 2r_{3,y}^i r_{3,z}^i)$$

7. Inertia tensor of the unit-mass mesh about the origin is computed as follows:

$$I_o = \sum_i I^{\triangle_i} V_\triangle^i$$

8. Inertia tensor about the center of mass is calculated using the parallel axis theorem:

$$I_{CoM} = \rho[I_o - V \cdot \cdot (\vec{r}_{CoM}^{\,2} I_{3\times 3} - \vec{r}_{CoM} \otimes \vec{r}_{CoM})]$$

where

$\rho = m/V$ is the density of the vessel (the software tool assumes uniform distribution of mass m), $I_{3\times3}$ is a 3-by-3 unit matrix and the $\otimes$ sign represents the outer vector product.

9. The principal axes of the body are found using the Jacobi eigenvalue algorithm $I_{CoM} = R I_{CoM}^{diag} R^T$, where

$$R = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix}$$

is the rotation matrix to the principal axes of the body (composed from the eigenvectors of I_{CoM}), and I_{CoM}^{diag} is the diagonalized inertia tensor (composed from the eigenvalues of I_{CoM}). The Jacobi iteration stops when all off-diagonal elements are less than the threshold value of 10^{-12} multiplied by the sum of the absolute values on the diagonal, or when the limit of 20 iterations is reached. It is possible to use this algorithm for inertia tensor diagonalization because I_{CoM} is a real symmetric matrix.

In the subsequent simulation, the origin of the local coordinate system of the vessel $\{X_s, Y_s, Z_s\}$ is always kept at the center of mass $\vec{r}_{CoM}$ and its orientation is kept equal to the principal axes orientation, such that the inertia tensor is constant and diagonal (equal to I_{CoM}^{diag}). The software ensures that both the fluid and the collision meshes are always synchronized with this coordinate system.

The simulator uses unit quaternions to represent rotations during the dynamical simulation process. Therefore, the rotation matrix to the principal axes R is converted to a unit quaternion $Q_{principal} = \{Q_x, Q_y, Q_z, Q_w\}$ using Algorithm 1. The algorithm ensures a non-degenerative production of a unit quaternion to represent rotation matrix R.

Initial position and orientation of the vessel in the simulator are given by the user in the input file as the 3D coordinate of the center of mass $\vec{r}_{CoM}^{\,init}$ in the global frame $\{X_g, Y_g, Z_g\}$ and 3 Cardan angles $\{r, p, q\}$ in radians. The software converts these angles into a unit quaternion as follows:

$$Q_{user} = (\sin\frac{r}{2}\cos\frac{p}{2}\cos\frac{y}{2} - \cos\frac{r}{2}\sin\frac{p}{2}\sin\frac{y}{2},$$
$$\cos\frac{r}{2}\sin\frac{p}{2}\cos\frac{y}{2} + \sin\frac{r}{2}\cos\frac{p}{2}\sin\frac{y}{2},$$
$$\cos\frac{r}{2}\cos\frac{p}{2}\sin\frac{y}{2} - \sin\frac{r}{2}\sin\frac{p}{2}\cos\frac{y}{2},$$
$$\cos\frac{r}{2}\cos\frac{p}{2}\cos\frac{y}{2} + \sin\frac{r}{2}\sin\frac{p}{2}\sin\frac{y}{2})$$

Then, initial quaternion of the simulated vessel is computed by applying the user-defined rotation to the principal axes of the body using quaternion multiplication operation:

Algorithm 1 Matrix-quaternion conversion algorithm

```
$T = R_{11} + R_{22} + R_{33}$
**if** $T > 0$ **then**
    $S = 2\sqrt{T+1}$
    $Q_x = -\frac{R_{32}-R_{23}}{S}$
    $Q_y = -\frac{R_{13}-R_{31}}{S}$
    $Q_z = -\frac{R_{21}-R_{12}}{S}$
    $Q_w = \frac{S}{4}$
**else**
    **if** $R_{11} > R_{22}\, and\, R_{11} > R_{33}$ **then**
        $S = 2\sqrt{1 + R_{11} - R_{22} - R_{33}}$
        $Q_x = \frac{S}{4}$
        $Q_y = \frac{R_{12}+R_{21}}{S}$
        $Q_z = \frac{R_{31}+R_{13}}{S}$
        $Q_w = -\frac{R_{32}-R_{23}}{S}$
    **else if** $R_{22} > R_{33}$ **then**
        $S = 2\sqrt{1 + R_{22} - R_{11} - R_{33}}$
        $Q_x = \frac{R_{12}+R_{21}}{S}$
        $Q_y = \frac{S}{4}$
        $Q_z = \frac{R_{23}+R_{32}}{S}$
        $Q_w = -\frac{R_{13}-R_{31}}{S}$
    **else**
        $Sc = 2\sqrt{1 + R_{33} - R_{11} - R_{22}}$
        $Q_x = \frac{R_{13}+R_{31}}{S}$
        $Q_y = \frac{R_{23}+R_{32}}{S}$
        $Q_z = \frac{S}{4}$
        $Q_w = -\frac{R_{21}-R_{12}}{S}$
    **end if**
**end if**
**Ensure:** $Q$ is normalized
```

$$\begin{aligned} Q_{init} &= Q_{user}Q_{principal} = Q_2Q_1 = \\ &= (Q_{1,w}Q_{2,x} + Q_{1,x}Q_{2,w} + Q_{1,y}Q_{2,z} - Q_{1,z}Q_{2,y}, \\ &\quad Q_{1,w}Q_{2,y} + Q_{1,y}Q_{2,w} + Q_{1,z}Q_{2,x} - Q_{1,x}Q_{2,z}, \\ &\quad Q_{1,w}Q_{2,z} + Q_{1,z}Q_{2,w} + Q_{1,x}Q_{2,y} - Q_{1,y}Q_{2,x}, \\ &\quad Q_{1,w}Q_{2,w} - Q_{1,x}Q_{2,x} - Q_{1,y}Q_{2,y} - Q_{1,z}Q_{2,z}) \end{aligned}$$

Finally, for free running simulation type, initial linear and angular velocities of the vessel are fetched from the input file and assigned to the simulated body in the selected physical engine.

4.2 Towing Carriage Creation

For an oblique towing simulation type the towing carriage is created by the software as a kinematic body in the physical engine, i.e. it is represented by an animated object that has a constant velocity irrespective of the forces acting on it (a moving body with infinite mass). Velocity of the carriage along the global X_g axis is fetched from the input file and assigned both to its kinematic body and to the body of the vessel, for initial synchronization (see Figure 4 where the carriage is represented by a dark-red rectangular cuboid positioned directly above the vessel). Towing carriage velocity is constant throughout the whole simulation process.

For visualization purposes the user can specify the height of the carriage $z^{user}_{carriage}$ in the input file. It will be represented by a cube located at $(r^{init}_{CoM,x}, r^{init}_{CoM,y}, z^{user}_{carriage})$ in the $\{X_g, Y_g, Z_g\}$ frame and moving with the desired velocity along the X_g axis. The total towing distance is specified by the user in the input file. When this distance is covered, the towing velocity is reset to zero, and the towing process is stopped.

The carriage and the vessel are connected to each other by a prismatic joint (Figure 6), which is a slider restricting 5 DOF of relative motion between the vessel and the carriage. Relative displacements only along the Z_g axis are allowed. The height of the joint z^{user}_{joint} is specified by the user in the input file, and its initial position in the simulation is set to $(r^{init}_{CoM,x}, r^{init}_{CoM,y}, z^{user}_{joint})$ in the $\{X_g, Y_g, Z_g\}$ frame. If the user does not specify z^{user}_{joint}, the height of the joint will be equal to $r^{init}_{CoM,z}$.

This arrangement ensures a physical towing process in the numerical simulation, meaning that the vessel is "dragged" by the towing carriage, and the resistance force can be measured by the prismatic joint.

4.3 Ice Field Generation

The ice field is created inside a rectangular region on the $\{X_g, Y_g\}$ plane according to the input file speci-

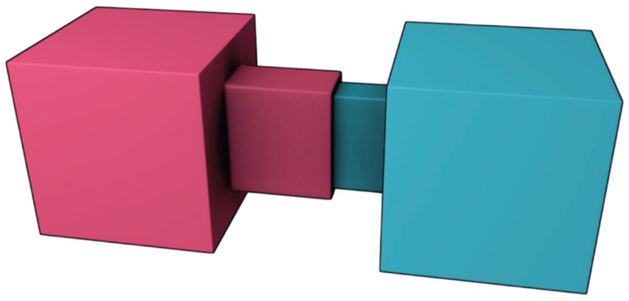

Figure 6: A horizontally aligned prismatic joint (Coumans, 2014).

fication. Before generating the ice field the software checks that the requested rectangular region of the ice field is fully contained inside the ice tank boundaries (Figure 11), so that no ice floes fall outside the actually simulated domain. Within the ice field rectangle the numerical tool can generate a level ice sheet (Section 4.3.1) or 3 different types of broken ice fields: randomly-distributed (Section 4.3.2), grid-distributed or size-distributed. The latter 2 are not discussed in this paper.

After the 2D ice field generation step is completed each ice floe polygon is positioned in the $\{X_g, Y_g\}$ plane. Then, all ice floes are created as dynamic bodies in the selected physical engine, i.e. their collision and fluid meshes are generated. Collision meshes are obtained by extruding the 2D polygonal shapes of the ice floes along the Z_g axis to obtain a uniform thickness h_i according to the input file specification. The result of the extrusion operation is a set of polyhedral rigid bodies with their side faces parallel to the Z_g axis (Figure 7). These polyhedra are used as collision meshes of the ice floes during the following dynamical simulation process.

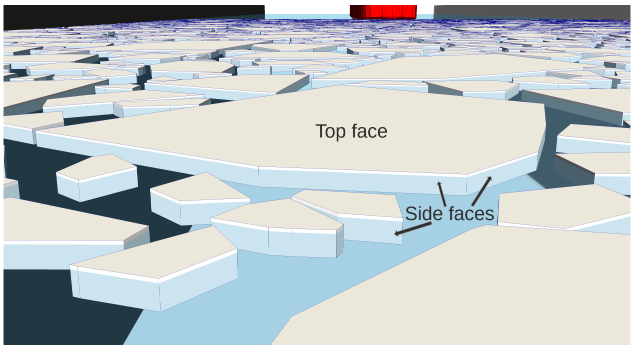

Figure 7: Ice floes in the virtual ice tank.

Fluid meshes of the ice floes are created differently for rectangular and polygonal geometries. Rectangular ice floe meshes are generated by manually triangulating their faces (Figure 8, a-d), while polygonal ice floes are first triangulated in-plane using a constrained Delaunay triangulation library Poly2Tri (Green, 2013): Figure 9, a-b. Then, the triangulated polygon is duplicated to create an extruded 3D prism of thickness h_i, which will constitute the top and bottom faces of the fluid mesh (Figure 9, c). Finally, the side faces of the polyhedron are triangulated to complete the mesh (Figure 9, d).

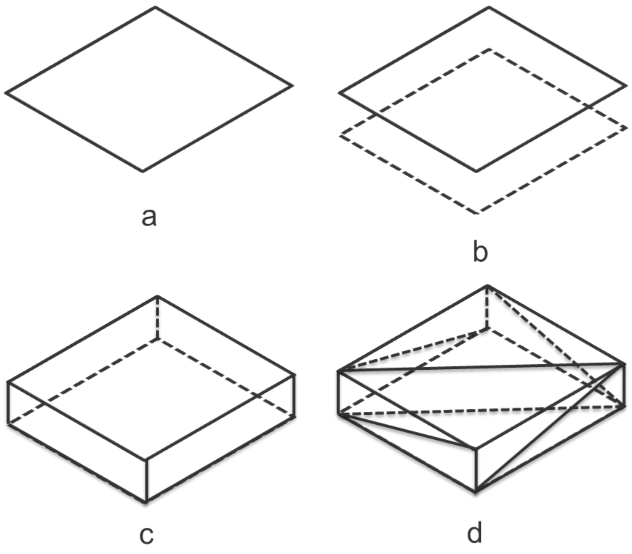

Figure 8: Rectangular cuboid triangulation.

Inertial properties of the rectangular ice floes are computed using analytical formulae. The mass is calculated as follows: $m_i = L_i W_i h_i \rho_i$, where ρ_i is the ice density. Inertia tensor about the principal axes is computed as follows:

$$I_{box} = m_i/12 \begin{pmatrix} L_i^2 + h_i^2 & 0 & 0 \\ 0 & W_i^2 + h_i^2 & 0 \\ 0 & 0 & L_i^2 + W_i^2 \end{pmatrix}$$

The principal axes themselves are parallel to the sides of the rectangular cuboid (Figure 10). The origin of the principal axes is located at the center of mass, which in this case coincides with the geometrical center (red dot in Figure 10).

Inertial properties of polyhedral ice floes are calculated using their fluid meshes and the same techniques as for the vessel (Section 4.1, steps 1-9). The only difference is that in step 8 of the algorithm the ice density is used instead of the structural density (it is

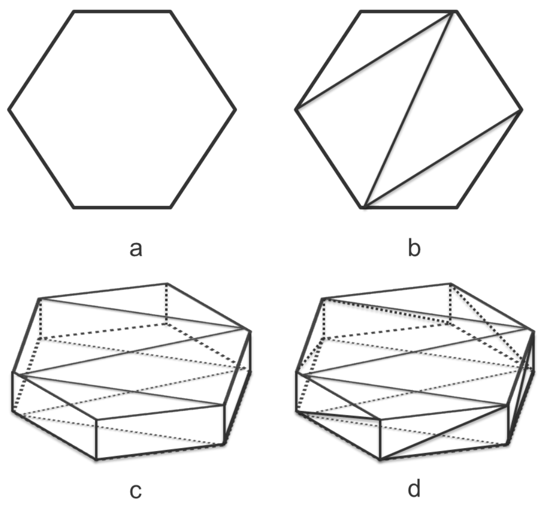

Figure 9: Polyhedron triangulation.

assumed that the ice floes have uniform density and their mass can be computed as $m_i = A_{2D} h_i \rho_i$, where A_{2D} is the area of the ice floe polygon). Otherwise, the exact same source code is used for the ice floes and the vessel. Furthermore, in the same manner as for the vessel, the simulator ensures that the origin of the local coordinate system of the ice floes $\{X_i, Y_i, Z_i\}$ is always kept at the center of mass, and its orientation is always kept equal to the principal axes, such that the inertia tensor in the local frame remains constant and diagonal throughout the whole simulation process.

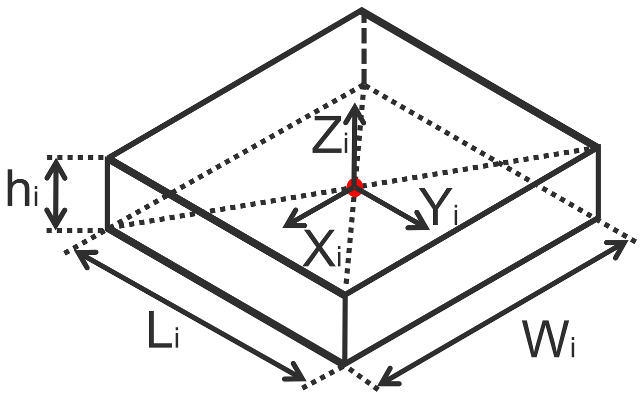

Figure 10: Dimensions of a rectangular cuboid.

The software initializes the vertical positions of the ice floes such that their draft is equal to $h_i \frac{\rho_i}{\rho_w}$, where ρ_w is the water density. Therefore, at the start of the simulation process all ice floes are created in equilibrium with the fluid. Additionally, the ice field generation algorithm ensures that the ice floes do not intersect each other and do not physically interact with each other at the start of the simulation process (Section 4.3.2).

To create the ice floes in the physical engine through the PAL interface, their initial positions and orientations have to be converted into PAL 4×4 matrices. To achieve this, first, the orientation quaternion of an ice floe is generated from its axis-angle representation: $Q = (k_x \sin\frac{\alpha}{2}, k_y \sin\frac{\alpha}{2}, k_z \sin\frac{\alpha}{2}, \cos\frac{\alpha}{2})$. Since $k_x = k_y = 0, k_z = 1$, this expression simplifies to $Q = (0, 0, \sin\frac{\alpha}{2}, \cos\frac{\alpha}{2})$, where α is the rotation angle of the ice floe around the Z_g axis. Then, a 3×3 rotation matrix is generated from the quaternion:

$$R = \begin{pmatrix} 1 - 2\frac{Q_y^2+Q_z^2}{\|Q\|^2} & 2\frac{Q_xQ_y+Q_wQ_z}{\|Q\|^2} & 2\frac{Q_xQ_z-Q_wQ_y}{\|Q\|^2} \\ 2\frac{Q_xQ_y-Q_wQ_z}{\|Q\|^2} & 1 - 2\frac{Q_x^2+Q_z^2}{\|Q\|^2} & 2\frac{Q_yQ_z+Q_wQ_x}{\|Q\|^2} \\ 2\frac{Q_xQ_z+Q_wQ_y}{\|Q\|^2} & 2\frac{Q_yQ_z-Q_wQ_x}{\|Q\|^2} & 1 - 2\frac{Q_x^2+Q_y^2}{\|Q\|^2} \end{pmatrix}$$

where $\|Q\|^2 = Q_x^2 + Q_y^2 + Q_z^2 + Q_w^2$. Finally, the 4×4 transformation matrix for the PAL interface is computed as follows:

$$T = \begin{pmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ r_x & r_y & r_z & 1 \end{pmatrix}$$

where $\vec{r} = (r_x, r_y, r_z)^T$ is the 3D position vector of the ice floe's center of mass. The resulting matrix T is used for initializing every ice floe in the selected physical engine.

4.3.1 Level Ice Field Generation

The level ice field is generated as a rectangular region extruded along the Z_g axis and uniformly filled with the ice material of thickness h_i. In the physical engine it is represented by a static rectangular cuboid which is not affected by forces from any other objects in the simulation domain (same rigid body type as the ice tank walls). However, the level ice itself can break and can produce reaction forces on the vessel and the dynamic ice pieces in contact with it. The ice force computation algorithms are described in Section 5.

4.3.2 Randomly-distributed Ice Field Generation

This generation algorithm can produce 2 different ice field types: rectangular and polygonal, depending on the shape of the individual ice floes selected by the user in the input file. The rectangular ice field consists of rectangles, while the polygonal ice field consists of convex polygons. The actual packing algorithm is the same for both ice field types. Furthermore, both ice field types are characterized by the same floe size parameters in the input file: the maximal floe size and the minimal floe size. They are defined as the length and the width of bounding rectangles in the $\{X_g, Y_g\}$ plane: $\vec{d}_{min} = (l_{min}, w_{min})$ and $\vec{d}_{max} = (l_{max}, w_{max})$ (Figure 11).

Before the algorithm is executed, 4 artificial rectangular boundaries with thickness b are created around the ice field rectangle to provide a "safety" margin for preventing polygon creation on the border of the region (Figure 11). These boundaries are used exclusively by the ice field generation algorithm, and they are not created in the physical simulation.

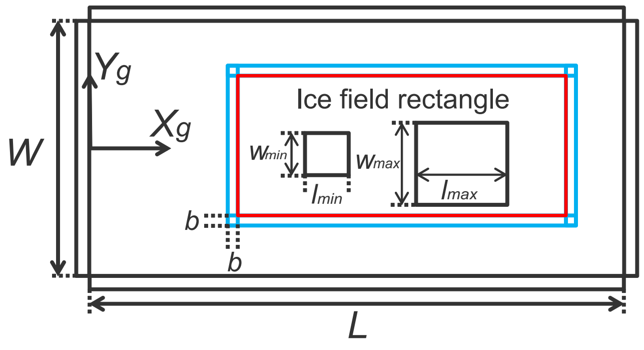

Figure 11: The ice field rectangle and its "safety" margins.

The actual generation algorithm starts by subdividing the user-defined floe size interval into N_{size} sub-intervals (Figure 12), and the packing procedure starts with the largest floes: $[\vec{d}^{cur}_{min}, \vec{d}^{cur}_{max}] = [\vec{d}_{min} + (N_{size} - 1)\frac{\vec{d}_{max} - \vec{d}_{min}}{N_{size}}, \vec{d}_{max}]$. Without subdivision into sub-intervals the algorithm tends to produce ice floes with almost exclusively $\vec{d}_{min}$ and $\vec{d}_{max}$ sizes, which results in unnaturally looking ice fields. Moreover, if the bounds of the sub-intervals are too narrow with respect to d_{lim}, which is the maximal allowed floe sides ratio (floe narrowness threshold), they are immediately corrected according to Algorithm 2 in order to avoid creation of unphysically narrow ice floes.

Algorithm 2 Interval narrowness correction.

```
if $\frac{d^{cur}_{min,y}}{d^{cur}_{min,x}} > d_{lim}$ then $d^{cur}_{min,y} = d_{lim} d^{cur}_{min,x}$
end if
if $\frac{d^{cur}_{max,y}}{d^{cur}_{max,x}} > d_{lim}$ then $d^{cur}_{max,y} = d_{lim} d^{cur}_{max,x}$
end if
if $\frac{d^{cur}_{max,x}}{d^{cur}_{max,y}} > d_{lim}$ then $d^{cur}_{min,x} = d_{lim} d^{cur}_{min,y}$
end if
if $\frac{d^{cur}_{min,x}}{d^{cur}_{max,y}} > d_{lim}$ then $d^{cur}_{max,x} = d_{lim} d^{cur}_{max,y}$
end if
```

Then, a candidate ice floe polygon is created. In case of the rectangular ice field type, the candidate polygon is created as a centered axis-aligned rectangle with size $(d^{cur}_{min,x}, d^{cur}_{min,y})$. In case of the polygonal ice field type, the candidate polygon is created as shown in Figure 13. First, a unit square is created and a small space t is reserved on all of its edges next to the vertices (Figure 13, a). Then, 4 random points are generated on the remaining space of the 4 edges (Figure 13, b). These 4 points constitute the vertices of the initial polygon, i.e. a quadrilateral. Next, one of the 4 vertices of the quadrilateral is selected randomly and a small space t is reserved on the 2 edges adjacent to it (Figure 13, c. Blue point is the one selected). Then, 2 more points are created on the remaining space of these 2 edges (Figure 13, c. Green points are the ones generated). Finally, the old point is discarded (the blue one in Figure 13, c), and the 2 new points are added to the polygon (the green ones in Figure 13, c), making it a pentagon (Figure 13, d). This procedure of selecting a random vertex and adding 2 new ones instead of it is repeated as many times as needed to create an N_{ver}-gon, where $N_{ver} \geq 4$ is the amount of vertices required in the final polygon. As an example, Figure 13 shows the creation of a heptagon, i.e. a 7-vertex polygon, using the outlined procedure. An important property of the algorithm is that it ensures creation of convex polygons, which makes them suitable for subsequent usage in the physical engine.

After the vertices of the polygon are created, its axis-aligned bounding box is obtained by looping through all of the vertices and finding their minimal and maximal coordinates: $x_{min}, x_{max}, y_{min}, y_{max}$ in the $\{X_g, Y_g\}$ plane. Then, a scaling vector is introduced: $\vec{d}_{scale} = (\frac{d^{cur}_{min,x}}{x_{max} - x_{min}}, \frac{d^{cur}_{min,y}}{y_{max} - y_{min}})$ and each vertex of the polygon $\vec{r}_i$ is centered and scaled using this vector to ensure user-defined proportions along the X_g and Y_g axes:

$$\vec{r}_i \leftarrow \vec{r}_i \cdot \vec{d}_{scale} - 0.5\vec{d}^{cur}_{min} - (x_{min}, y_{min})^T \cdot \vec{d}_{scale}$$

Finally, the polygon is either expanded or contracted in order to match the required area $d^{cur}_{min,x} \cdot d^{cur}_{min,y}$. In order to do this, the current area of the polygon is first calculated as follows:

$$A_{polygon} = 0.5 \cdot (r_{N_{ver},x} r_{1,y} - r_{N_{ver},y} r_{1,x} + \\ + \sum_{i=1}^{N_{ver}-1} r_{i,x} r_{i+1,y} - r_{i,y} r_{i+1,x})$$

Finally, the coordinate vector of each vertex in the polygon is scaled as follows:

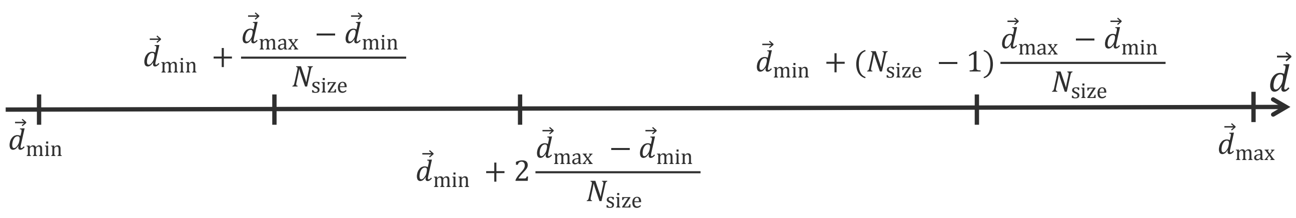
$\vec{d}_{\min} + \frac{\vec{d}_{\max} - \vec{d}_{\min}}{N_{\text{size}}}$
$\vec{d}_{\min} + (N_{\text{size}} - 1)\frac{\vec{d}_{\max} - \vec{d}_{\min}}{N_{\text{size}}}$
$\vec{d}$
$\vec{d}_{\min}$
$\vec{d}_{\min} + 2\frac{\vec{d}_{\max} - \vec{d}_{\min}}{N_{\text{size}}}$
$\vec{d}_{\max}$

Figure 12: User-defined floe size interval with subdivision.

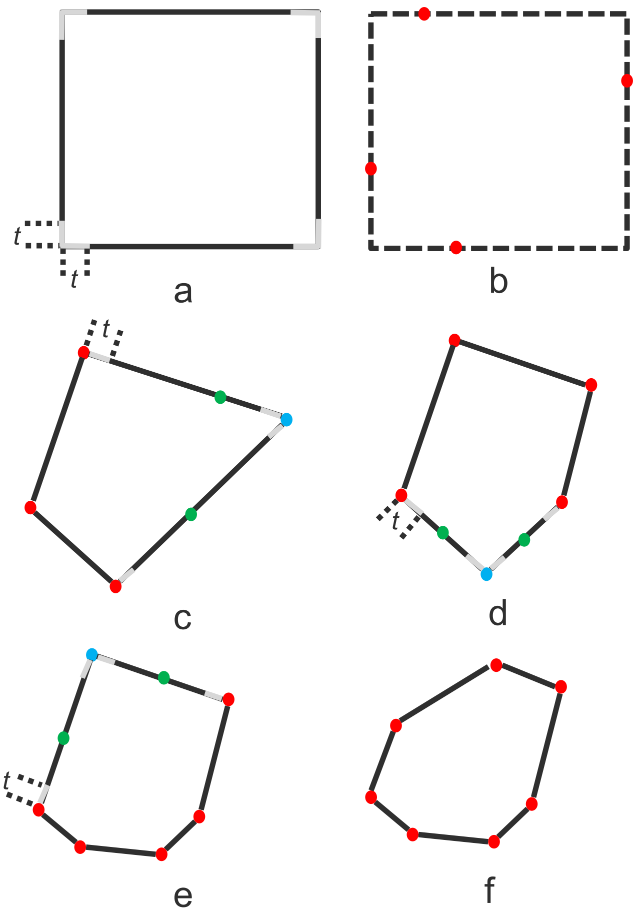

Figure 13: Random polygon generation procedure.

$$\vec{r}_i \leftarrow \vec{r}_i \cdot \sqrt{\frac{d^{cur}_{min,x} \cdot d^{cur}_{min,y}}{A_{polygon}}}$$

The polygonal ice floe generation algorithm, described in the previous paragraphs, performs random vertex selection and polygon point creation using a pseudo-random number generator. This generator is used here, as well as in other parts of the software, to produce pseudo-random integers and real numbers using a user-defined seed from the input file (an unsigned integer number). A custom-made routine, initialized by the seed, creates pseudo-random numbers in order to ensure consistency and repeatability of the simulations. All random numbers in the software have uniform distributions in their respective ranges.

After the candidate ice floe polygon is finally created, the ice floe placement routine is initiated in a loop. Firstly, a random 2D displacement vector $\vec{r}_{loc}$ inside the ice field rectangle and a random rotation angle α in the range $[0, \pi]$ are applied to each vertex $\vec{r}_i$ of the candidate polygon:

$$\begin{pmatrix} r_{i,x} \\ r_{i,y} \end{pmatrix} \leftarrow \begin{pmatrix} r_{i,x}\cos\alpha - r_{i,y}\sin\alpha + r_{loc,x} \\ r_{i,x}\sin\alpha + r_{i,y}\cos\alpha + r_{loc,y} \end{pmatrix}$$

Then, the software checks if the candidate polygon intersects any of the previously added polygons or the ”safety” margins around the ice field rectangle. The Boost.Geometry library (Gehrels et al., 2014) is used for the intersection testing (the *intersects* function), and the axis-aligned bounding boxes of the polygons are checked first, before the actual polygons, catering for increased numerical efficiency of the packing routine.

If no intersections are found the packing algorithm makes N_{expand} attempts to enlarge the candidate polygon using Algorithm 3 (bisection method). The algorithm finds the maximal possible scaling factor $\vec{d}_{best}$ which produces no intersections with the previously added polygons or the ”safety” margins around the ice field rectangle.

Then, the packing routing makes N_{trials} attempts to slightly move, rotate and enlarge the candidate polygon

Algorithm 3 Polygon expansion procedure.

```
$\vec{d}_{low} = (1,1)^T$
$\vec{d}_{high} = (d^{cur}_{max,x}/d^{cur}_{min,x}, d^{cur}_{max,y}/d^{cur}_{min,y})^T$
**for** $i = 1 : N_{expand}$ **do**
    $\vec{d}_{test} = (\vec{d}_{low} + \vec{d}_{high})/2$
    $\vec{r}_i \leftarrow \vec{r}_i \cdot \vec{d}_{test}$
    **if** intersects other polygons or margins **then**
        $\vec{d}_{high} = \vec{d}_{test}$
    **else**
        $\vec{d}_{low} = \vec{d}_{test}$
    **end if**
**end for**
Maximal expansion $\vec{d}_{best} = \vec{d}_{low}$
```

in order to find its optimal placement and size in a separate loop (the trials loop). Finally, the algorithm adds the polygon to the ice field and checks if the target ice concentration has been reached. The full ice field generation routine is outlined in Figure 14.

The main limitation of the algorithm is that, due to its stochastic nature, it practically cannot produce ice concentrations above 80%. Nevertheless, the software reports the actually achieved ice concentration to the user at the end of the 2D generation process, so that it can be compared to the target concentration. Additionally, the ice field generation time is reported in textual format.

The final step before the physical creation of an ice floe is an update of its 2D polygon vertices in the $\{X_g, Y_g\}$ plane:

$$\begin{pmatrix} r_{i,x} \\ r_{i,y} \end{pmatrix} \leftarrow$$
$$\leftarrow \begin{pmatrix} r_{i,x}d_{best,x}\cos\alpha_{best} - r_{i,y}d_{best,y}\sin\alpha_{best} + r^{best,x}_{loc} \\ r_{i,x}d_{best,x}\sin\alpha_{best} + r_{i,y}d_{best,y}\cos\alpha_{best} + r^{best,y}_{loc} \end{pmatrix}$$

Figure 15 shows an example of the generated rectangular and polygonal 57 × 10 m ice fields with 70% ice concentration and the following generation parameters: $\vec{d}_{min} = (0.1, 0.1)$ m, $\vec{d}_{max} = (0.5, 0.5)$ m, $b = 1$ m, $N_{size} = 3$, $d_{lim} = 3$, $N_{ver} = 8$, $t = 0.1$ m, $N_{fails} = 1024$, $j^{init}_{pos} = 0.1$, $j^{init}_{rot} = 0.2$, $N_{expand} = 7$ and $N_{trials} = 64$. The generation time of the rectangular ice field was 8.73 s, while for the polygonal ice field it took 11.86 s.

5 Simulation Loop

The simulation loop of the numerical tool is centered around the physical engine software. Although the application supports several different engines, the ice

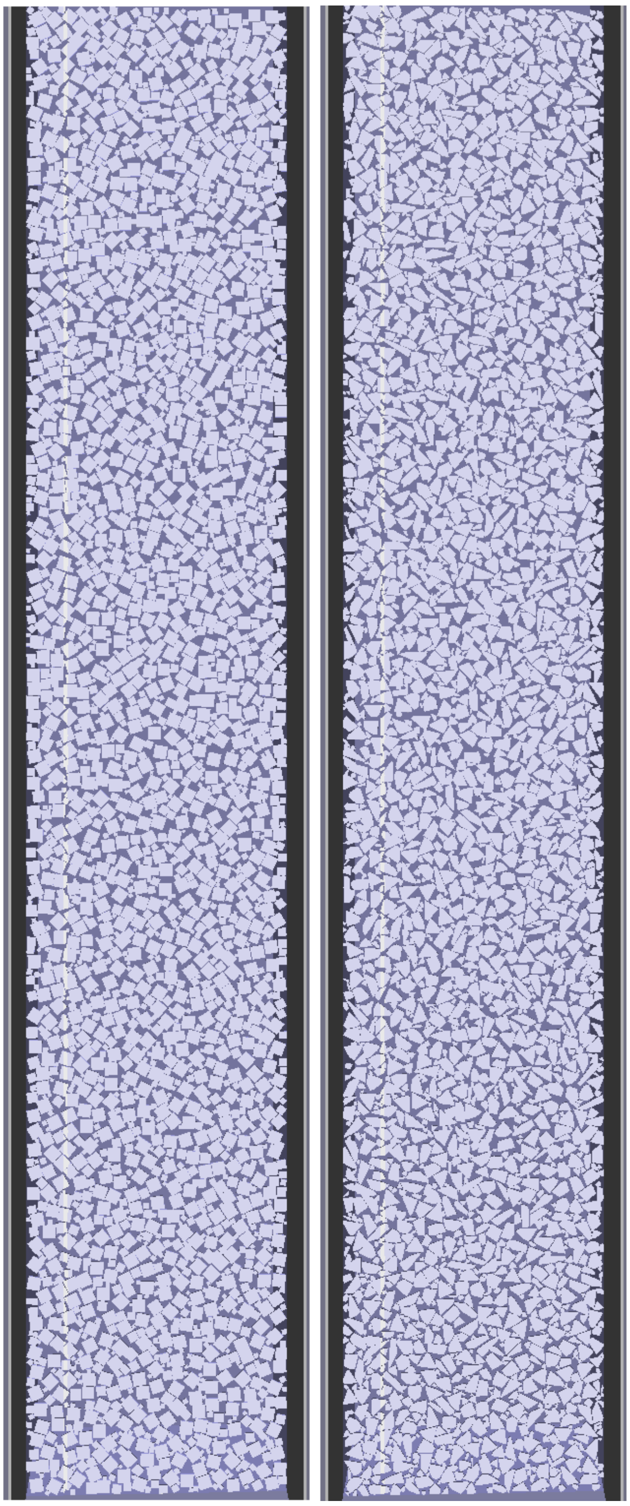
Figure 15: Two different 10 × 57 m ice fields generated by the software: rectangular (left) and polygonal (right).

```
Initialize ice field area $A_{field}^{total} = 0$
for $i = 1 : N_{size}$ do
    $\vec{d}_{min}^{cur} = \vec{d}_{min} + (N_{size} - i)\frac{\vec{d}_{max} - \vec{d}_{min}}{N_{size}}$
    $\vec{d}_{max}^{cur} = \vec{d}_{min} + (N_{size} - i + 1)\frac{\vec{d}_{max} - \vec{d}_{min}}{N_{size}}$
    Execute Algorithm 2
    Create a candidate ice floe polygon
    for $j = 1 : N_{fails}$ do
        Create random $\vec{r}_{loc}$ and $\alpha$
        Apply $\vec{r}_{loc}$ and $\alpha$ to the candidate polygon
        if Candidate polygon intersects any other polygons or "safety" margins then
            $j++$
        else
            $j = 1$
            $\vec{r}_{loc}^{best} = \vec{r}_{loc}$
            $\alpha_{best} = \alpha$
            Find $\vec{d}_{best}$ using Algorithm 3
            $j_{pos} = max(d_{min,x}^{cur}, d_{min,y}^{cur}) \cdot j_{pos}^{init}$
            $j_{rot} = \pi \cdot j_{rot}^{init}$
            for $k = 1 : N_{trials}$ do
                $\vec{r}_{loc}^{trial} = \vec{r}_{loc} + \vec{r}_{rand}(-j_{pos}, j_{pos})$
                $\alpha_{trial} = \alpha + rand(-j_{rot}, j_{rot})$
                Apply $\vec{r}_{loc}^{trial}$ and $\alpha_{trial}$ to the candidate polygon
                if Candidate polygon intersects any other polygons or "safety" margins then
                    $k++$
                else
                    Find $\vec{d}_{trial}$ using Algorithm 3
                    if $|\vec{d}_{trial}| > |\vec{d}_{best}|$ then
                        $\vec{r}_{loc}^{best} = \vec{r}_{loc}^{trial}$
                        $\alpha_{best} = \alpha_{trial}$
                        $\vec{d}_{best} = \vec{d}_{trial}$
                    end if
                end if
            end for
            $A_{floe} = A_{polygon} \cdot d_{best,x} \cdot d_{best,y}$
            $A_{field}^{upcoming} = A_{field}^{total} + A_{floe}$
            if $A_{field}^{upcoming} > A_{field}^{target}$ then
                $A_{floe}^{corrected} = A_{floe} - (A_{field}^{upcoming} - A_{field}^{target})$
                if $A_{floe}^{corrected} > 10^{-4}\ m^2$ then
                    $\vec{d}_{best} \leftarrow \vec{d}_{best} \cdot \sqrt{A_{floe}^{corrected}/A_{floe}}$
                else
                    Algorithm end
                end if
                Physically create the ice floe
                Algorithm end
            end if
            Physically create the ice floe
            $A_{field}^{total} \leftarrow A_{field}^{total} + A_{floe}$
        end if
    end for
end for
```

Figure 14: Randomly-distributed ice field generation algorithm.

breaking functionality is currently implemented only in the Bullet-based version of the model. Therefore, this Section describes the simulation loop as implemented in the Bullet engine.

One iteration of the simulation loop, which is progressing with a fixed time step specified by the user in the input file, is shown in Figure 16. Before the loop actually starts, the physical properties of all simulated bodies (i.e. the vessel, the ice floes and the walls of the ice tank) are created in the computer memory. For each body with index i these properties are: the dynamical type of the body (static, kinematic or dynamic), its mass m_i and inertia tensor I_i, its collision and fluid meshes, its position $\vec{r}_i$ and orientation quaternion Q_i, its linear and angular velocities $\vec{v}_i, \vec{\Omega}_i$ and the forces and torques $\vec{f}_i, \vec{\tau}_i$ acting on it ($i = 1 : N_{body}$ and N_{body} is the total amount of simulated bodies). If the towing carriage is present in the simulation, its prismatic joint is mapped into memory as well.

The following sub-sections describe the various steps of the simulation loop which are executed sequentially at every time step.

5.1 Collision Detection

The first step of the simulation loop is execution of the collision detection routine. It uses the collision meshes of the objects to compute their geometrical intersections. First, the axis-aligned bounding boxes (AABBs) are computed for each simulated body in the global $\{X_g, Y_g, Z_g\}$ frame. As an early filter of potentially colliding bodies, intersecting AABBs provide a list of object pairs for further processing in the broad phase step of the collision detection routine. If the collision mesh of a body has a convex decomposition, a dynamic AABB tree is constructed for the body as a bounding volume hierarchy (BVH) in its local coordinate system. These BVHs accelerate the collision detection process by performing a middle phase step which uses a very fast insertion, removal and update of nodes and exploits the temporal coherence of the simulated scene. The broad phase and middle phase steps find all potentially overlapping pairs of objects and reduce the time complexity of the subsequent narrow phase contact detection step from $O(N_{body}^2)$ to $O(N_{body})$. The narrow phase step itself computes the contact manifolds on the overlapping geometries. Every contact manifold consists of one or more contact points which contain information about the two bodies in contact, the positions of the two closest (witness) points on each of the overlapping bodies, the contact normal vector and the contact distance along the normal vector (which is positive if the objects are separate and negative if there is a penetration). Later in the simulation loop these contact points are converted into contact constraints which ensure non-penetration and frictional forces between the simulated bodies. Therefore, the contact points also contain information about the friction and restitution coefficients between the colliding bodies. The narrow phase collision detection algorithm uses the Bullet’s implementation of the Separating Axis Theorem (SAT), because the default Gilbert-Johnson-Keerthi distance algorithm was found to be unstable in generating contact normals. Furthermore, the collision margins of the SAT algorithm were set to 0 in order to avoid contact point generation without the actual contacts between the bodies (this value was non-zero in the default version of the Bullet library). Finally, the value of the contact breaking threshold was set to 10^{-10} m, because the default value was leading to collision reporting of separated bodies.

5.2 Ice Crushing Constraint

The next step of the simulation loop, after the narrow phase collision detection is completed, is the ice crushing constraint formulation. This constraint allows coupling the multibody solver of the physical engine to the mechanical properties of the ice material, and eventually produces a more realistic physical behavior of the numerical model as reported in Scibilia et al. (2014). Technically, this step of the simulation loop is implemented as a collision system callback in the Bullet library.

All ice floes in the simulation domain are divided into 2 groups: breakable and unbreakable. Floes with waterplane areas $A_{floe} > L_{breakable} h_i$ are considered breakable. In the current implementation of the numerical model the value of $L_{breakable}$ can be easily adjusted in the simulator’s source code. Every breakable ice floe in contact with the vessel is processed to formulate the ice crushing constraint, while the unbreakable floes are not modified.

The ice crushing constraint formulation is a stepwise process which is executed for all breakable ice floes in contact with the vessel. The final objective of this process is to compute the list of geometrical intersections between the structural mesh and the ice floe mesh (Section 5.2.1), and the associated list of contact points from the physical engine (Section 5.2.2).

5.2.1 Geometrical Intersection Computation

The fluid mesh of the vessel is used to compute the geometrical intersection between the structure and the ice floes. First, each vertex i of the mesh is transformed into the coordinate system of the top face of the currently processed ice floe (Figure 7): $\vec{r}_i^{floe} = R_{floe}^T(\vec{r}_i - \vec{r}_{floe}^{COM}) - (0, 0, h_i/2)^T$, where R_{floe} is the rotation matrix of the ice floe, $\vec{r}_{floe}^{COM}$ is the position

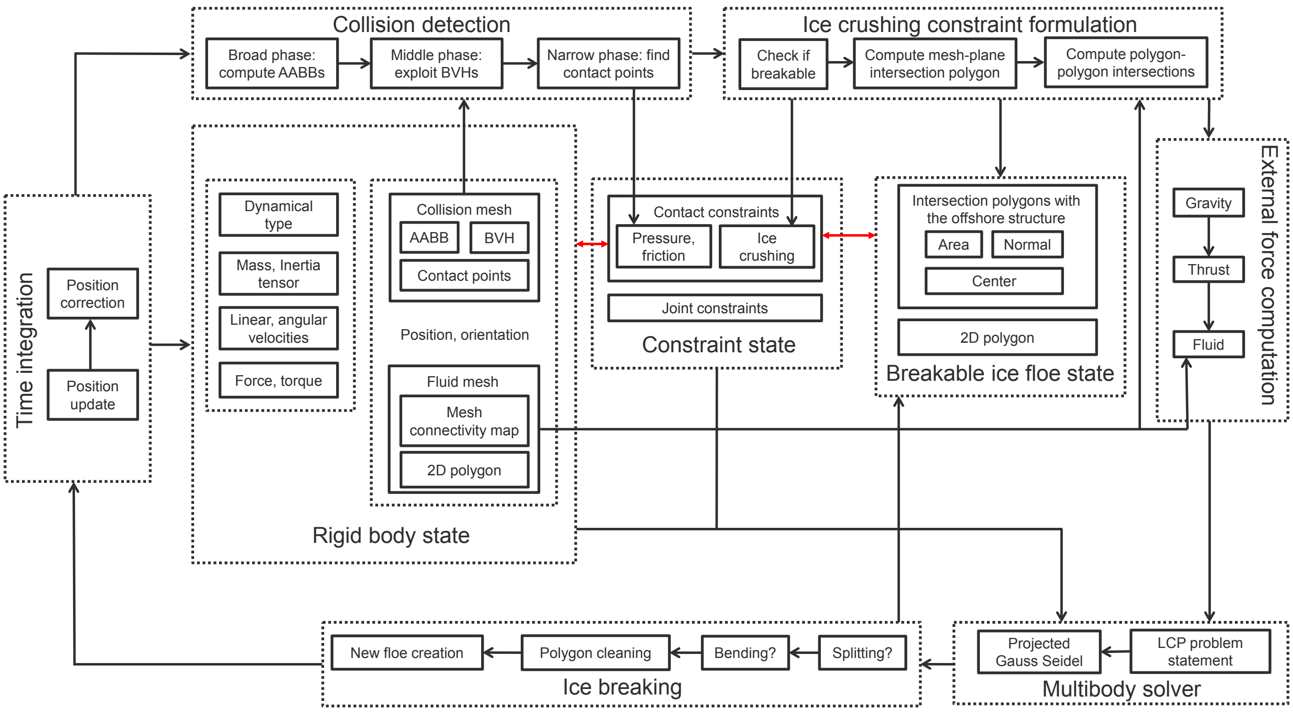

Figure 16: One iteration of the simulation loop.

of the floe's center of mass in the global frame, $\vec{r}_i$ is the position of vertex i of the fluid mesh in the global frame, $\vec{r}_i^{\,floe}$ is the position of vertex i of the fluid mesh in the ice floes frame and h_i is the ice thickness. All subsequent calculations in this Section are performed in this coordinate system, i.e. the frame of the top face of a breakable ice floe.

For every edge of the vessels mesh, the intersection point between that edge and the plane of the floe's top face is found using Algorithm 4. After the first intersecting edge is found, each of the 2 edges in the 2 adjacent triangles to the first intersecting edge are checked for intersection with the plane. The mesh connectivity map is exploited for this task: for every edge of the mesh this map contains information about the indices of the 2 adjacent triangular faces and which other 2 edges those faces contain. This allows "propagating" the edge-plane intersection contour through the whole mesh in an efficient manner, until a closed 2D polygon is formed. To avoid numerical errors, for every new candidate point of the intersection polygon the algorithm computes the distance from that point to the previous point already added to the polygon. If that distance is below 10^{-12} m, the algorithm does not add the new point to the polygon.

After the intersection contour is computed, every point of the resulting polygon is complemented with information about its 3D normal vector. This vector

Algorithm 4 Edge-plane intersection test.

```
Edge vertices in 3D: $\vec{r}_1, \vec{r}_2$
if $r_{1,z} \cdot r_{2,z} > 0$ then
    No intersection
end if
if $|r_{1,z}| < 10^{-8}$ m and $|r_{2,z}| < 10^{-8}$ m then
    No intersection
end if
Intersection point $\vec{r}_{int} = \vec{r}_1 + \frac{|r_{1,z}|}{|r_{1,z}|+|r_{2,z}|}(\vec{r}_2 - \vec{r}_1)$
```

$\vec{n}_\triangle$ is computed as the normal to the triangular face that intersects the floe's top face plane.

The full mesh-plane intersection algorithm is provided in Figure 17. This algorithm can produce several intersection polygons between the floe's top face plane and the mesh of the vessel, for example when there are thruster boxes in the waterplane of the vessel (Figure 18). In this case, the list of all intersection polygons between the vessel and the ice floes will be denoted $\{S_{polygon}\}$ in the remainder of this paper.

Next, for every polygon from the $\{S_{polygon}\}$ set, the software computes intersections with the currently processed breakable ice floe polygon. The *intersection* function of the Boost.Geometry library (Gehrels et al., 2014) is used for this purpose. The result of this computation is a list of intersection polygons between the

Figure 18: An example of a possible set of intersection polygons between the floe’s top face plane and the mesh of the vessel.

ice floe and the vessel $\{I_{polygon}\}$.

Then, for every polygon from the $\{I_{polygon}\}$ set, the 3D normal vectors are computed as follows. First, every edge of the polygon is checked for equality with all edges of the intersection polygons from the $\{S_{polygon}\}$ list. To do that, Algorithm 5 is executed for both vertices of every edge of the polygon. If both distances, reported by Algorithm 5, are below 10^{-8} m, the edges are stated to coincide. In this case, the normal of the first vertex of the currently processed edge $\vec{n}_{\triangle_i}$ is added into an accumulative vector $\vec{n}_{ice} \leftarrow \vec{n}_{ice} + \vec{n}_{\triangle_i}$ for the currently processed polygon. After all coinciding edges are found and their normals are accumulated, the resulting unit normal of the polygon is computed as follows: $\vec{n}_{polygon} = -(\vec{n}_{ice}/|\vec{n}_{ice}|)$, where $|\vec{n}_{ice}|$ is the length of the $\vec{n}_{ice}$ vector. The negation sign is used here to ensure that the vector is pointing from the ice floe into the vessel, which is needed for subsequent ice fracture computations in Section 5.5. If the result of this operation is a vector with negative Z_g component, i.e. pointing from the structure into the ice, the whole

```
Mark all triangles in the mesh as *unprocessed*
for All edges in the mesh do
    for All edges in the mesh do
        if 2 triangles adjacent to the current edge are unprocessed then
            Try to find edge-plane intersection $\vec{r}_{int}$ using Algorithm 4 and terminate the loop if found
        end if
    end for
    if Edge-plane intersection is found then
        for All edges in the mesh do
            for 2 triangles adjacent to the current edge do
                if Current triangle is unprocessed then
                    for 2 edges in the current triangle (not the current edge) do
                        Try to find edge-plane intersection using Algorithm 4
                        if Edge-plane intersection found then
                            Mark the intersection point as the *next intersection point*
                            Mark the intersecting edge as the *next edge*
                            Mark the current triangle as $\triangle_n$
                        end if
                    end for
                    Mark the current triangle as *processed*
                    Exit the loop if edge-plane intersection was found
                end if
            end for
            Add $r_{int}^{x,y}$ point to the intersection polygon and assign $\vec{n}_{\triangle_n} = ||(\vec{r}_2 - \vec{r}_1) \times (\vec{r}_3 - \vec{r}_1)||$ to this point
            Go to the *next edge* and the *next intersection point*
        end for
    end if
end for
```

Figure 17: Mesh-plane intersection computation algorithm.

polygon is discarded as invalid. The same happens if the algorithm fails to produce an accumulated $\vec{n}_{ice}$ vector.

Algorithm 5 Distance from a point to an edge in 2D.

```
Input point $\vec{r}$
Edge vertices $\vec{r}_1, \vec{r}_2$
$\vec{d} = \frac{\vec{r}_2 - \vec{r}_1}{|\vec{r}_2 - \vec{r}_1|}$
**if** $\vec{d} \cdot (\vec{r} - \vec{r}_1) < 0$ **then**
    Return $|\vec{r} - \vec{r}_1|$
**else if** $\vec{d} \cdot (\vec{r} - \vec{r}_1) > |\vec{r}_2 - \vec{r}_1|$ **then**
    Return $|\vec{r} - \vec{r}_2|$
**else**
    Return $|(-d_y, d_x)^T \cdot (\vec{r} - \vec{r}_1)|$
**end if**
```

Finally, the areas of the valid polygons are computed using the $A_{polygon}$ formula presented previously and stored for subsequent ice fracture computations in Section 5.5. Furthermore, the axis-aligned bounding boxes of these polygons are obtained by looping through all of their vertices and finding their minimal and maximal coordinates: $x_{min}, x_{max}, y_{min}, y_{max}$ in the $\{X_g, Y_g\}$ plane. Then, the centers of the bonding boxes are found: $x_{center} = 0.5(x_{min} + x_{max}), y_{center} = 0.5(y_{min} + y_{max})$ and saved as the vertical load application points $\vec{r}^{\,vert}_{load}$ for each polygon. These values are used in the following step of the simulation loop.

5.2.2 Contact Point Association

The goal of this step is to associate every contact point, generated by the narrow phase collision detection system, with a specific floe-vessel intersection polygon from the $\{I_{polygon}\}$ list. To accomplish that, each contact point is first computed as an average value of the witness points on the overlapping bodies (found in Section 5.1): $\vec{r}_{contact} = 0.5(\vec{r}^{\,1}_{contact} + \vec{r}^{\,2}_{contact})$. Then, these contact points are transformed into the local frame of the floe's top face (i.e. the same frame as of the floe-structure intersection polygons): $\vec{r}^{\,floe}_{contact} = R^T_{floe}(\vec{r}_{contact} - \vec{r}^{\,COM}_{floe}) - (0, 0, h_i/2)^T$. Next, the 2D X, Y positions of these points are found by discarding their z-coordinates: $\vec{r}^{\,2D}_{contact} = (r_{contact,x}, r_{contact,y})^T$. Finally, the minimal distances from these 2D points to the vertical load application points of every floe-structure intersection polygon are found: $min|\vec{r}^{\,2D}_{contact} - \vec{r}^{\,vert}_{load}|$, and the contact points themselves are associated to the polygons with the minimal distance values. In this way every contact constraint gets associated to one floe-structure intersection polygon from the $\{I_{polygon}\}$ list.

The final step of the ice crushing constraint formulation process is the computation of the ice crushing limit for every contact point:

$$p_{lim} = \frac{\triangle t \gamma \sigma_c A_{polygon}}{max(0.05, n_{polygon,z} N^{inters}_{contact})}$$

where $\triangle t$ is the time step, γ is a uniformly-distributed random value in the range $[0.5 - 1.0]$ which accounts for contact surface imperfections, σ_c is the compressive strength of the ice and $N^{inters}_{contact}$ is the amount of contact points associated to the current intersection polygon from the $\{I_{polygon}\}$ list. Technically, p_{lim} is an upper limit value for the normal contact constraint impulse, expressed in $N \cdot s$. It contains a cut-off constant 0.05 to avoid degenerative contact conditions (i.e. division by zero in the above stated formula). It means that the numerical model bypasses the handling of ice crushing by vertically-sided vessels, which is a major limitation of the simulator.

At the end of the ice crushing constraint formulation process every breakable ice floe in the simulation has an association to its physical body (through PAL), a 2D polygon of its top face surface, and a list of 2D intersection patches between that polygon and the structural mesh: $\{I_{polygon}\}$. Finally, every 2D intersection patch has a list of contact points from the physical engine associated to it, and every contact point knows its ice crushing constraint limit. This relationship is illustrated in Figure 16 with red arrows, and it is one of distinct differences of the current model from that of Lubbad and Løset (2011), where ice crushing forces are calculated explicitly and applied directly to the vessel at every time step.

5.3 External Force Computation

The first external force applied to all simulated objects is the gravity force: $\vec{f}_{grav} = m\vec{g}$, where m is the mass of the object and $\vec{g}$ is the 3D gravity vector defined by the user in the input file. This force is applied to the center of mass of every simulated object.

The second external force is the thruster force. It is applied to the vessel in free running and DP simulation modes. In the free running mode, both the thruster force $\vec{f}_{thrust}$ and the thruster torque $\vec{\tau}_{thrust}$ are fetched from the user input file, while in the DP mode these loads are acquired from the DP interface. In both modes the user specifies the force and torque in the local frame of the vessel $\{X_s, Y_s, Z_s\}$, so they have to be transformed into the global frame for subsequent usage in the multibody solver (Section 5.4): $R^T \vec{f}_{thrust}, R^T \vec{\tau}_{thrust}$, where R is the rotation matrix of the vessel at the current time step. Both the force and the torque are applied to the center of mass of the vessel.

5.3.1 Fluid Force Computation

As specified in Section 4, the fluid domain in the numerical model is represented by a static half-space demarcated by the $\{X_g, Y_g\}$ plane, i.e. the water level is always at $Z_g = 0$. Free surface effects are not simulated, and no volumetric deformations of the fluid are allowed. Only buoyancy and drag forces are applied to the objects, and the whole underwater domain has a constant and uniform density. The fluid meshes of the objects are used for the fluid force computations in the software.

Currently 2 different types of the mathematical fluid model are supported by the software: engine-specific and internal. The engine-specific method is implemented only in the Vortex engine and, therefore, does not support ice fracturing. This method is described in Metrikin et al. (2013b). The drag coefficient for the Vortex model is provided by the user in the input file, while the drag torque and added mass effects are disabled in the engine through the PAL interface. This method can be used for simulating vessels in unbreakable ice.

The algorithm of the internal mathematical fluid model supports ice fracturing. It is based on the method and source code of Catto (2006) which clips the triangles of the fluid mesh against the water plane and computes the submerged volume V_{sub} of the mesh and its centroid $\vec{r}_c$ using contributions from the individual triangles. However, 2 additional quantities are computed in the current numerical model for every submerged triangle in order to calculate the drag force: its area $A_\triangle$ and the linear velocity projection $v^\triangle_{proj}$. To do that, the following auxiliary vector is found first: $\vec{r}_{cross} = (\vec{r}_2 - \vec{r}_1) \times (\vec{r}_3 - \vec{r}_1)$, where $\vec{r}_{1,2,3}$ are 3D vertex coordinates of the triangle. Then, the area of the triangle is found as follows: $A_\triangle = 0.5\sqrt{r^2_{cross,x} + r^2_{cross,y} + r^2_{cross,z}}$. To find the linear velocity projection $v^\triangle_{proj}$, the body's velocity is first transformed from the global frame into the local frame of the object: $\vec{v}_{loc} = R\vec{v}$, where $\vec{v}$ is the linear velocity of the body in the global frame and R is the rotation matrix of the body. Then, the velocity projection onto the current triangle is found as follows:

$$v^\triangle_{proj} = 0.5\frac{\vec{v}_{loc}}{\sqrt{v^2_{loc,x} + v^2_{loc,y} + v^2_{loc,z}}} \cdot \vec{r}_{cross}$$

if $v^\triangle_{proj} < 0$, it is reset to 0, so that the drag force is not applied to shaded surfaces. The total submerged surface area of the object is then found as $A_{sub} = \sum A_\triangle$, and the total velocity projection on the submerged surface of the body is $v_{proj} = \sum(v^\triangle_{proj} A_\triangle)$, where summation is performed over all submerged triangles of the mesh.

The buoyancy force is computed as follows: $\vec{f}_{buoy} = -\vec{g}\rho_w V_{sub}$. Then, velocity of the center of buoyancy is computed as follows: $\vec{v}_c = \vec{\Omega} \times (\vec{r}_c - \vec{r}_{CoM}) + \vec{v}$, where $\vec{\Omega}$ is the angular velocity of the body in the global frame. Finally, the drag force is found as follows:

$$\vec{f}_{drag} = -0.5\rho_w \vec{v}_c (C_d v_{proj} + \\ + \sqrt{v^2_{c,x} + v^2_{c,y} + v^2_{c,z}} C_s A_{sub})$$

where C_d is the form drag coefficient and C_s is the skin friction coefficient from the input file. The total fluid force is then found as follows: $\vec{f}_{fluid} = \vec{f}_{buoy} + \vec{f}_{drag}$.

The fluid torque can be computed in 2 different ways, depending on the user's choice. The first option is the model of Catto (2006). It requires the average length of the mesh which is computed as follows. First, the axis-aligned bounding box is found in the local frame of the mesh: $x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}$. Then, the average length of the mesh is found as follows: $l_{avg} = \frac{1}{3}(x_{max} - x_{min} + y_{max} - y_{min} + z_{max} - z_{min})$. Finally, the drag torque is computed as follows:

$$\vec{\tau}_{drag} = -m\frac{V_{sub}}{V}\vec{\Omega} C_a l^2_{avg}$$

where V is the total volume of the fluid mesh and $C_a[1/s]$ is the angular drag coefficient from the input file.

The second option for the fluid torque computation is the following: $\vec{\tau}_{drag} = -0.5\vec{\Omega}^2 \rho_w C_a A_{sub} l^3_{avg}$, which is supposed to approximate the viscous drag torque. The angular drag coefficient C_a in this formula is non-dimensional, in contrast to the one used in the expression of Catto (2006).

Finally, the total fluid torque is computed as follows:

$$\vec{\tau}_{fluid} = \vec{\tau}_{drag} + (\vec{r}_c - \vec{r}_{CoM}) \times \vec{f}_{fluid}$$

the second term appears because the point of the drag force application is assumed to coincide with the center of buoyancy.

Although the numerical model does not account for wind actions, the above described method can, in principle, be applied for wind force and torque computations.

The final external force applied to each body in the simulation domain is equal to $\vec{f}_{ext} = \vec{f}_{grav} + R^T \vec{f}_{thrust} + \vec{f}_{fluid}$, and the external torque is equal to $\vec{\tau}_{ext} = R^T \vec{\tau}_{thrust} + \vec{\tau}_{fluid}$. These forces and torques are acting only on dynamic bodies in the simulation, i.e. the static and kinematic objects are not affected.

5.4 Multibody Solver

After all external forces are computed and the ice crushing constraints are formulated the multibody solving step is performed according to the procedure described in this Section.

First, the Newton-Euler equations for body i out of total N_{body} are formulated in the global inertial frame $\{X_g, Y_g, Z_g\}$ as follows:

$$m_i \frac{d}{dt}\vec{v}_i = \vec{f}_i$$

$$I_i \frac{d}{dt}\vec{\Omega}_i + \vec{\Omega}_i \times I_i\vec{\Omega}_i = \vec{\tau}_i$$

where

$$\vec{f}_i = \vec{f}_{ext,i} + \sum_j \vec{f}_{cont,j} + \sum_k \vec{f}_{joint,k}$$

and

$$\vec{\tau}_i = \vec{\tau}_{ext,i} + \sum_j \vec{r}_{i,j} \times \vec{f}_{cont,j} + \sum_k \vec{r}_{i,k} \times \vec{f}_{joint,k}$$

where $\vec{f}_{cont,j}$ is the contact force on body i from body j, $\vec{f}_{joint,k}$ is the joint force on body i from joint k, $\vec{r}_{i,j}$ is the vector from body's i center of mass to the witness point of contact j and $\vec{r}_{i,k}$ is the vector from body's i center of mass to the force application point of joint k. $I_i = RI_i^{body}R^T$ is the inertia tensor of body i in the global frame, obtained by transforming the diagonal inertia tensor I_i^{body} from the principal axes of the body using its rotation matrix R.

The Newton-Euler equations can be written in matrix form:

$$M_i \frac{d\vec{u}_i}{dt} = \vec{F}_i$$

where

$$M_i = \begin{pmatrix} m_i I_{3\times3} & 0 \\ 0 & I_i \end{pmatrix} \in \mathbb{R}^{6\times6}$$

is the generalized mass matrix of body i, $\vec{u}_i = (\vec{v}_i^T, \vec{\Omega}_i^T)^T \in \mathbb{R}^{6\times1}$ is the generalized velocity vector of body i and $\vec{F}_i = (\vec{f}_i^T, (\vec{\tau}_i - \vec{\Omega}_i \times I_i\vec{\Omega}_i)^T)^T \in \mathbb{R}^{6\times1}$ is the generalized force vector of body i.

From classical mechanics principle of virtual work it is known that the generalized joint force on body i in a single constraint k can be expressed as follows: $\vec{F}_{joint,k} = J_{joint,ik}^T\vec{\lambda}_{joint,k}$, where $\vec{\lambda}_{joint,k} \in \mathbb{R}^{m\times1}$ is a vector of Lagrange multipliers which account for the reaction forces coming from the joint bearings. These multipliers can take any real value, both positive and negative. The dimension m of the Lagrange multiplier vector is equal to the amount of degrees of freedom removed by the joint. For example, the prismatic joint removes 5 degrees of freedom, so $m = 5$. $J_{joint,ik} \in \mathbb{R}^{m\times6}$ is the joint constraint Jacobian which ensures satisfaction of the locking condition for the connected bodies i and j: $(J_{joint,ik}, J_{joint,jk})\vec{u}_k = 0$, where $\vec{u}_k = (\vec{u}_i^T, \vec{u}_j^T)^T \in \mathbb{R}^{12\times1}$ is the velocity of joint k. For example, the prismatic joint connecting the towing carriage to the vessel in the numerical model is a slider along the Z_g axis, and its Jacobians are (according to Chapter 4.9.3 in Erleben (2005)):

$$J_{joint,ik} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & x_{axis,x} & x_{axis,y} & x_{axis,z} \\ 0 & 1 & 0 & y_{axis,x} & y_{axis,y} & y_{axis,z} \end{pmatrix}$$

$$J_{joint,jk} = \begin{pmatrix} 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & x_{axis,x} & x_{axis,y} & x_{axis,z} \\ 0 & -1 & 0 & y_{axis,x} & y_{axis,y} & y_{axis,z} \end{pmatrix}$$

where $\vec{x}_{axis} = 0.5(\vec{r}_i - \vec{r}_j) \times (1, 0, 0)^T$ and $\vec{y}_{axis} = 0.5(\vec{r}_i - \vec{r}_j) \times (0, 1, 0)^T$

Contact forces can also be expressed using the Jacobian notation. The generalized contact force on body i from body j can be expressed as follows: $\vec{F}_{cont,j} = J_{cont,ij}^T\vec{\lambda}_{cont,j}$, where $\vec{\lambda}_{cont,j} \in \mathbb{R}^{3\times1}$ is the vector of Lagrange multipliers which account for the normal pressure and frictional forces in contact j and $J_{cont,ij}$ is the contact constraint Jacobian:

$$J_{cont,ij} = \begin{pmatrix} -\vec{n}_{ij}^T & -(\vec{r}_{i,j} \times \vec{n}_{ij})^T \\ -\vec{t}_{ij,1}^T & -(\vec{r}_{i,j} \times \vec{t}_{ij,1})^T \\ -\vec{t}_{ij,2}^T & -(\vec{r}_{i,j} \times \vec{t}_{ij,2})^T \end{pmatrix} \in \mathbb{R}^{3\times6}$$

where $\vec{n}_{ij} \in \mathbb{R}^{3\times1}$ is the contact normal, reported by the narrow phase collision detection system; $\vec{t}_{ij,1}, \vec{t}_{ij,2} \in \mathbb{R}^{3\times1}$ are 2 orthogonal vectors in the tangential contact plane, produced by the narrow phase collision detection system for the respective contact point. According to the Newton's third law, the constraint Jacobian for body j in the same contact is:

$$J_{cont,ji} = \begin{pmatrix} \vec{n}_{ij}^T & (\vec{r}_{j,i} \times \vec{n}_{ij})^T \\ \vec{t}_{ij,1}^T & (\vec{r}_{j,i} \times \vec{t}_{ij,1})^T \\ \vec{t}_{ij,2}^T & (\vec{r}_{j,i} \times \vec{t}_{ij,2})^T \end{pmatrix} \in \mathbb{R}^{3\times6}$$

Complementarity condition for the normal contact force states that the normal contact force can be positive only if the normal contact velocity is zero, and vice versa:

$$(J_{cont,ij}\vec{u}_i)_x \geq 0 \perp 0 \leq (\vec{\lambda}_{cont,j})_x \leq \frac{p_{lim,j}}{\triangle t}$$

where $p_{lim,j}$ is the ice crushing constraint limit for the normal pressure force in contact $i \Leftrightarrow j$. If the contact does not have an assigned ice crushing constraint limit (e.g. a contact between unbreakable ice floes), the limit value is set to: $p_{lim,j} = +\infty$.

Complementarity condition for the frictional force direction states that the frictional force shall be opposite to the sliding velocity:

$$\lambda_{aux,j} + (J_{cont,ij}\vec{u}_i)_y \geq 0 \perp (\vec{\lambda}_{cont,j})_y \geq 0$$

$$\lambda_{aux,j} + (J_{cont,ij}\vec{u}_i)_z \geq 0 \perp (\vec{\lambda}_{cont,j})_z \geq 0$$

where $\lambda_{aux,j}$ is an auxiliary Lagrange multiplier without a real physical meaning, but an approximation to the magnitude of the relative tangential velocity in contact j.

Finally, complementarity condition for the frictional force magnitude (Coulomb dry friction law) states that the frictional force cannot be greater than the normal force multiplied by the friction coefficient μ:

$$\mu(\vec{\lambda}_{cont,j})_x - (\vec{\lambda}_{cont,j})_y \geq 0 \perp \lambda_{aux,j} \geq 0$$

$$\mu(\vec{\lambda}_{cont,j})_x - (\vec{\lambda}_{cont,j})_z \geq 0 \perp \lambda_{aux,j} \geq 0$$

where it is assumed that $\mu_s = \mu_k = \mu$ is the friction coefficient for all contacts in the multibody system.

Now it is possible to state the full system equations in matrix form:

$$M\frac{d\vec{u}}{dt} = \vec{F}$$

where

$$M = \begin{pmatrix} M_1 & 0 & \cdots & 0 \\ 0 & M_2 & \cdots & 0 \\ \cdots & \cdots & \ddots & \vdots \\ 0 & \cdots & \cdots & M_{N_{body}} \end{pmatrix} \in \mathbb{R}^{6N_{body} \times 6N_{body}}$$

is the inertia matrix of the N_{body}-system;

$$\vec{u} = (\vec{u}_1^T, \vec{u}_2^T, \cdots, \vec{u}_{N_{body}}^T)^T \in \mathbb{R}^{6N_{body} \times 1}$$

is the system velocity vector;

$$\vec{F} = \vec{F}_{ext} + J_{joint}^T\vec{\lambda}_{joint} + J_{cont}^T\vec{\lambda}_{cont} \in \mathbb{R}^{6N_{body} \times 1}$$

is the system force vector, where

$$\vec{F}_{ext} = (\vec{f}_1^T, (\vec{\tau}_1 - \vec{\Omega}_1 \times I_1\vec{\Omega}_1)^T)^T, \\ \vec{f}_2^T, (\vec{\tau}_2 - \vec{\Omega}_2 \times I_2\vec{\Omega}_2)^T)^T, \cdots, \vec{f}_{N_{body}}^T, \\ (\vec{\tau}_{N_{body}} - \vec{\Omega}_{N_{body}} \times I_{N_{body}}\vec{\Omega}_{N_{body}})^T)^T \in \mathbb{R}^{6N_{body} \times 1}$$

is the external force vector of the system,

$$J_{joint} = \\ \begin{pmatrix} J_{joint,11} & J_{joint,21} & \cdots & J_{joint,N_{body}1} \\ J_{joint,12} & J_{joint,22} & \cdots & J_{joint,N_{body}2} \\ \cdots & \cdots & \ddots & \vdots \\ J_{joint,1N_{joint}} & J_{joint,2N_{joint}} & \cdots & J_{joint,N_{body}N_{joint}} \end{pmatrix} \\ \in \mathbb{R}^{K \times 6N_{body}}$$

is the system joint Jacobian, where N_{joint} is the total amount of joints in the system and K is the total amount of degrees of freedom restricted by the joints. In the current numerical model, $N_{joint} = 1$ and $K = 5$, so the J_{joint} matrix is extremely sparse with only 2 sub-matrices $J_{joint,jk}$ being non-zero (the ones for the vessel and the towing carriage). $\vec{\lambda}_{joint} = (\vec{\lambda}_{joint,1}^T, \vec{\lambda}_{joint,2}^T, ..., \vec{\lambda}_{joint,N_{joint}}^T)^T \in \mathbb{R}^{K \times 1}$ is the system vector of Lagrange multipliers which account for joint reaction forces.

$$J_{cont} = \\ \begin{pmatrix} J_{cont,11} & J_{cont,21} & \cdots & J_{cont,N_{body}1} \\ J_{cont,12} & J_{cont,22} & \cdots & J_{cont,N_{body}2} \\ \cdots & \cdots & \ddots & \vdots \\ J_{cont,1N_{cont}} & J_{cont,2N_{cont}} & \cdots & J_{cont,N_{body}N_{cont}} \end{pmatrix} \\ \in \mathbb{R}^{3N_{cont} \times 6N_{body}}$$

is the system contact Jacobian, where N_{cont} is the total amount of contacts in the system. J_{cont} is also extremely sparse, because all $J_{cont,ii}$ are equal to zero (the body cannot be in contact with itself) and every row of the matrix corresponds to only one contact which involves just 2 bodies, i.e. only two $J_{cont,ij}$ and $J_{cont,ji}$ sub-matrices are non-zero in each row. $\vec{\lambda}_{cont} = (\vec{\lambda}_{cont,1}^T, \vec{\lambda}_{cont,2}^T, ..., \vec{\lambda}_{cont,N_{cont}}^T)^T \in \mathbb{R}^{3N_{cont} \times 1}$ is the system vector of Lagrange multipliers which account for contact forces.

Joint lock condition on the system level can be stated simply as $J_{joint}\vec{u} = 0$, while complementarity condition for the normal contact forces in the whole system can be expressed as follows:

$$(J_{cont}\vec{u})_n \geq 0 \perp 0 \leq (\vec{\lambda}_{cont})_n \leq \frac{\vec{p}_{lim}}{\triangle t}$$

where $\vec{p}_{lim} = (p_{lim,1}, p_{lim,2}, \cdots, p_{lim,N_{cont}})^T \in \mathbb{R}^{N_{cont}\times 1}$ is the system vector of the ice crushing constraints and $(\vec{\lambda}_{cont})_n \in \mathbb{R}^{N_{cont}\times 1}$ are the normal contact forces.

Complementarity condition for the frictional force direction on the system level can be expressed as follows:

$$E\vec{\lambda}_{aux} + (J_{cont}\vec{u})_t \geq 0 \perp (\vec{\lambda}_{cont})_t \geq 0$$

where $\vec{\lambda}_{aux} = (\lambda_{aux,1}, \lambda_{aux,2}, \cdots, \lambda_{aux,N_{cont}})^T \in \mathbb{R}^{N_{cont}\times 1}$ is the system vector of auxiliary Lagrange multipliers accounting for the sliding velocity magnitude;

$$E = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 \\ 0 & \cdots & 0 & 1 \end{pmatrix} \in \mathbb{R}^{2N_{cont}\times N_{cont}}$$

is an auxiliary unit matrix, and $(\vec{\lambda}_{cont})_t \in \mathbb{R}^{2N_{cont}\times 1}$ are the frictional forces in the system. Finally, complementarity condition for the frictional force magnitude on the system level can be written as follows:

$$F(\vec{\lambda}_{cont})_n - E^T(\vec{\lambda}_{cont})_t \geq 0 \perp \vec{\lambda}_{aux} \geq 0$$

where

$$F = \begin{pmatrix} \mu & 0 & \cdots & 0 \\ 0 & \mu & \cdots & 0 \\ \cdots & \cdots & \ddots & \vdots \\ 0 & \cdots & 0 & \mu \end{pmatrix} \in \mathbb{R}^{N_{cont}\times N_{cont}}$$

is the friction coefficient matrix of the multibody system.

To formulate the solving process, the system acceleration vector is approximated as follows:

$$M\frac{\vec{u}^{new} - \vec{u}^{old}}{\triangle t} = \vec{F}_{ext} + J_{joint}^T\vec{\lambda}_{joint} + J_{cont}^T\vec{\lambda}_{cont}$$

The new velocity vector can be expressed as follows:

$$\vec{u}^{new} = \vec{u}^{old} + \\ + M^{-1}\triangle t(\vec{F}_{ext} + J_{joint}^T\vec{\lambda}_{joint} + J_{cont}^T\vec{\lambda}_{cont})$$

Inserting the last equation into the joint lock condition gives:

$$J_{joint}[\vec{u}^{old} + M^{-1}\triangle t\cdot \\ \cdot(\vec{F}_{ext} + J_{joint}^T\vec{\lambda}_{joint} + J_{cont}^T\vec{\lambda}_{cont})] = 0$$

Rearranging the terms on both sides of the equation leads to:

$$J_{joint}M^{-1}J_{joint}^T\vec{p}_{joint} + J_{joint}M^{-1}J_{cont}^T\vec{p}_{cont} + \\ + J_{joint}(\vec{u}^{old} + \triangle tM^{-1}\vec{F}_{ext}) = 0$$

where $\vec{p}_{joint} = \triangle t\vec{\lambda}_{joint}$ is the joint impulse vector and $\vec{p}_{cont} = \triangle t\vec{\lambda}_{cont}$ is the contact impulse vector. Similarly, complementarity condition for the normal contact forces can be written as follows:

$$(J_{cont}M^{-1}J_{joint}^T\vec{p}_{joint})_n + (J_{cont}M^{-1}J_{cont}^T\vec{p}_{cont})_n + \\ + [J_{cont}(\vec{u}^{old} + \triangle tM^{-1}\vec{F}_{ext})]_n \geq 0$$

and for the frictional force direction:

$$(J_{cont}M^{-1}J_{joint}^T\vec{p}_{joint})_t + (J_{cont}M^{-1}J_{cont}^T\vec{p}_{cont})_t + \\ + [J_{cont}(\vec{u}^{old} + \triangle tM^{-1}\vec{F}_{ext})]_t + E\vec{\lambda}_{aux} \geq 0$$

Finally, the full complementarity conditions of the system can be written in matrix form as shown in Figure 19. This mathematical formulation is a linear complementarity problem (LCP) of the following form:

$$A\vec{x} + \vec{b} \geq 0 \perp \vec{x}_{low} \leq \vec{x} \leq \vec{x}_{high}$$

The software framework solves the LCP using the Projected Gauss-Seidel method of the Bullet engine (as implemented in the *btSequentialImpulseConstraint-Solver* class) described in Section 7.2 of Catto (2005) and Section 6 of Erleben (2005). This method satisfies the normal and frictional contact constraints together with the joint constraints in a sequential impulse loop. A limit of 10 iterations is placed on the solver in order to achieve a fast yet stable simulation results. All constraints are solved together, because satisfying one constraint might violate another constraint (Coumans, 2014). Once the solving is finished, the software saves the obtained impulses $\vec{p}_{joint}$ and $\vec{p}_{cont}$, because they are re-used at the next simulation step as the initial guess of the Projected Gauss-Seidel method (which implements the so-called "warm-starting" of the solver).

The unknown impulses obtained by the LCP solver are used to compute the updated velocities of all objects in the system under assumption of no restitution ($e = 0$), i.e. all collisions are treated as inelastic:

$$\vec{u}^{new} = \vec{u}^{old} + \\ + M^{-1}\triangle t(\vec{F}_{ext} + J_{joint}^T\frac{\vec{p}_{joint}}{\triangle t} + J_{cont}^T\frac{\vec{p}_{cont}}{\triangle t})$$

The sleeping and damping features of the physical engine, as well as stiffness and damping in constraints (k_s, k_d), are deactivated for all bodies in the numerical model in order to avoid compromising accuracy of the simulations.

5.5 Ice Fracturing

After all forces and updated velocities of the objects in the system are found by the multibody solver, the ice fracturing step is performed.

5.5.1 Fracture Initiation

As described in Section 5.2, at the end of the ice crushing constraint formulation process every breakable ice floe has an associated list of 2D intersection patches between itself and the structural mesh. Furthermore, every intersection patch has an associated list of contact constraints from the physical engine (Figure 20). The multibody solver provides the forces in these contact constraints $\vec{f}_{cont}$, and the numerical model adds these forces together into the accumulative force vector exerted by the structure onto the ice patch: $\vec{f}_{ice} = \sum\limits_{patch} \vec{f}_{cont}$. The PAL interface is used to fetch the contact forces and to perform the summation independently in every patch. In the current version of the numerical model it is assumed that only the vertical Z_g-component of $\vec{f}_{ice}$ can lead to the global failure of a breakable ice floe, which means that it is controlled by initiation of radial cracks in the ice sheet (i.e. in-plane ice splitting failure is not considered).

The fracture initiation algorithm starts by checking if a radial crack can be nucleated at the current intersection patch. To perform the check, the breakable ice floe is idealized as a semi-infinite plate resting on an elastic foundation and subjected to a uniformly dis-

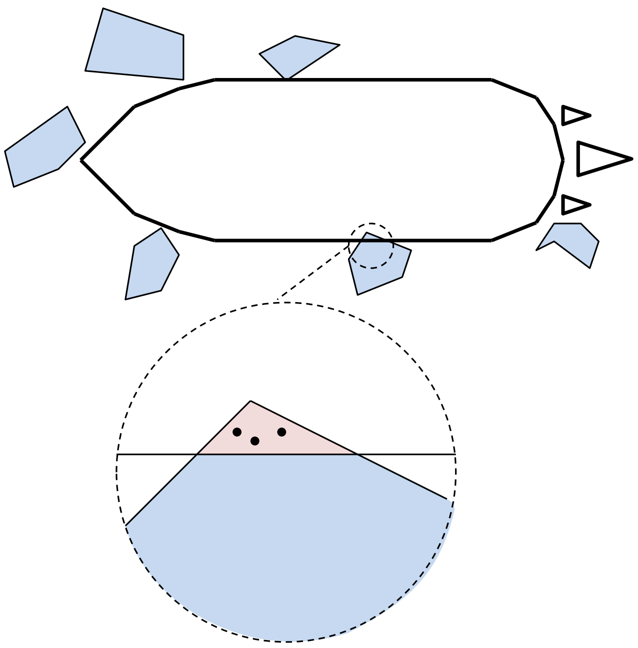

Figure 20: Intersection patch between a breakable ice floe and the vessel.

$$\underbrace{\begin{pmatrix} J_{joint}M^{-1}J_{joint}^T & (J_{joint}M^{-1}J_{cont}^T)_n & (J_{joint}M^{-1}J_{cont}^T)_t & 0 \\ (J_{cont}M^{-1}J_{joint}^T)_n & (J_{cont}M^{-1}J_{cont}^T)_n & 0 & 0 \\ (J_{cont}M^{-1}J_{joint}^T)_t & 0 & (J_{cont}M^{-1}J_{cont}^T)_t & E \\ 0 & F & -E^T & 0 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} \vec{p}_{joint} \\ (\vec{p}_{cont})_n \\ (\vec{p}_{cont})_t \\ \vec{\lambda}_{aux} \end{pmatrix}}_{\vec{x}} +$$

$$+ \underbrace{\begin{pmatrix} J_{joint}(\vec{u}^{\,old} + \triangle t M^{-1}\vec{F}_{ext}) \\ [J_{cont}(\vec{u}^{\,old} + \triangle t M^{-1}\vec{F}_{ext})]_n \\ [J_{cont}(\vec{u}^{\,old} + \triangle t M^{-1}\vec{F}_{ext})]_t \\ 0 \end{pmatrix}}_{\vec{b}} \geq 0 \perp$$

$$\perp \underbrace{\begin{pmatrix} -\infty \\ 0 \\ 0 \\ 0 \end{pmatrix}}_{\vec{x}_{low}} \leq \vec{x} \leq \underbrace{\begin{pmatrix} +\infty \\ \vec{p}_{lim} \\ +\infty \\ +\infty \end{pmatrix}}_{\vec{x}_{high}}$$

Figure 19: Complementarity conditions on system level in matrix form.

tributed vertical load in a semi-circular area (Lubbad and Løset, 2011). The radius of the loaded area is equal to $R_{load} = \sqrt{(2A_{patch})/\pi}$, and the areal density of the distributed load is equal to $q_{load} = f_{ice,z}/A_{patch}$, where A_{patch} is the area of the current intersection patch. The bending stress in the semi-infinite ice sheet σ_{yy} is calculated using the analytical solution derived in Section 6.5 of Lubbad and Løset (2011). If the value of this stress exceeds the flexural strength of the ice from the user input file: $\sigma_{yy} \geq \sigma_f$, a radial crack is nucleated at the point of the ice floe polygon which is closest to the vertical load application point $\vec{r}^{\,vert}_{load}$ of the current intersection patch (which was found in Section 5.2).

The starting point of the nucleated crack $\vec{r}^{\,start}_{crack}$ is found by iterating over all edges of the polygon using Algorithm 6 and finding the shortest distance to the vertical load application point $\vec{r}^{\,vert}_{load}$. Then, the potential crack propagation direction is found as follows: $\vec{n}_{break} = -\frac{(n_{polygon,x};n_{polygon,y})^T}{\sqrt{n^2_{polygon,x}+n^2_{polygon,y}}}$, where $\vec{n}_{polygon}$ is the polygon normal vector found in Section 5.2. Finally, in order to avoid incorrect geometrical intersection and difference computations later in the algorithm, the crack starting point is moved from the boundary of the polygon in the negative crack propagation direction by 0.1 mm: $\vec{r}^{\,start}_{crack} \leftarrow \vec{r}^{\,start}_{crack} - 0.0001 \cdot \vec{n}_{break}$.

Algorithm 6 Closest point to an edge in 2D.

```
Input point $\vec{r}$
Edge vertices $\vec{r}_1, \vec{r}_2$
$\vec{d} = \vec{r}_2 - \vec{r}_1$
if $|\vec{d}|^2 = d_x^2 + d_y^2 < 10^{-12}$ m then
    Return $\vec{r}_1$
end if
$t = [\vec{d} \cdot (\vec{r} - \vec{r}_1)]/|\vec{d}|^2$
if $t > 1$ then
    $t = 1$
else if $t < 0$ then
    $t = 0$
end if
Return $\vec{r}_1 + t(\vec{r}_2 - \vec{r}_1)$
```

Subsequent ice fracture computations require the amount of possible ice wedges produced by potential cracking of the breakable ice floe. This value is generated as a random variable N_{wedges} uniformly distributed between 3 and 5 (Lubbad and Løset, 2011; Nevel, 1992). Finally, the wedge angle is also needed for subsequent computations. It is found as follows: $\alpha_{wedge} = \pi/N_{wedges}$.

5.5.2 Ice Splitting Fracture

The polar angle of the first possible splitting crack θ_{crack} is found according to Algorithm 7 and assigned to the vector $\vec{n}_{split} = (\cos\theta_{crack}, \sin\theta_{crack})^T$. The other N_{wedges} possible splitting crack directions are computed by adding a step of α_{wedge} to the angle argument of $\vec{n}_{split}$.

Algorithm 7 Crack angle computation in 2D.

```
Input $\vec{n}_{break}$
$\theta_{crack} = atan2\frac{n_{break,y}}{n_{break,x}}$
if $\theta_{crack} < 0$ then
    $\theta_{crack} = \theta_{crack} + 2\pi$
end if
$\theta_{crack} = \theta_{crack} - 0.5\pi$
```

Testing of the splitting possibility is performed by shooting rays in the directions of every possible $\vec{n}_{split}$, except the ones pointing outside the ice floe polygon (Figure 21), and finding intersection points between the rays and the polygon using Algorithm 8 on every edge of the polygon. Degenerate geometrical conditions, when a splitting ray starts exactly from one of the polygon's vertices, are handled by discarding intersection points which are closer to each other than 10^{-6} m.

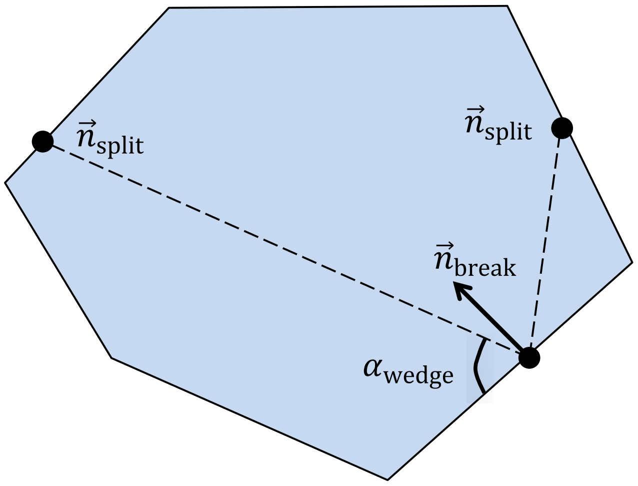

Figure 21: Possible splitting cracks in the ice floe polygon.

If more than 1 intersection point is found between the splitting ray and the ice floe polygon, the one closest to the crack starting point $\vec{r}^{\,start}_{crack}$ becomes the new crack starting point. Then, for all remaining intersection points $\vec{r}_{intersect}$ the following value is computed: $p = (\vec{r}_{intersect} - \vec{r}^{\,start}_{crack}) \cdot \vec{n}_{split}$. Intersection point with the minimal positive p is then becoming the ending point of the potential splitting crack $\vec{r}^{\,end}_{crack}$. This technique allows handling even concave ice floe polygons as shown in Figure 22.

Algorithm 8 Polygon edge - splitting ray intersection in 2D.

```
Polygon edge vertices $\vec{r}_1, \vec{r}_2$
$\delta = n_{split,x}(r_{1,y} - r_{2,y}) + n_{split,y}(r_{2,x} - r_{1,x})$
if $|\delta| < 10^{-8}$ m then
    Lines are parallel, no intersection
end if
$t = [(r_{1,x} - r^{start}_{crack,x})(r_{1,y} - r_{2,y})+$
$+(r_{1,y} - r^{start}_{crack,y})(r_{2,x} - r_{1,x})]/\delta$
if $t < -10^{-8}$ m or $t > 1 + 10^{-8}$ m then
    No intersection
end if
if $-10^{-8}m < t < 0$ then
    $t = 0$
end if
if $1 < t < 1 + 10^{-8}$ m then
    $t = 1$
end if
Return $\vec{r}_1 + t(\vec{r}_2 - \vec{r}_1)$
```

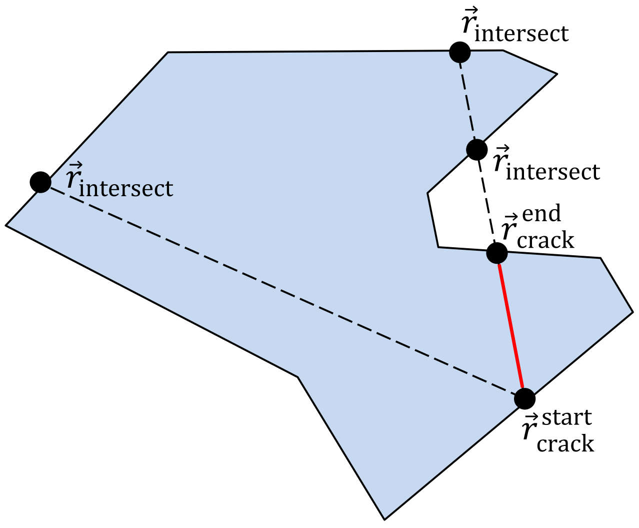
$\vec{r}_{\text{intersect}}$
$\vec{r}_{\text{intersect}}$
$\vec{r}_{\text{intersect}}$
$\vec{r}^{\text{end}}_{\text{crack}}$
$\vec{r}^{\text{start}}_{\text{crack}}$

Figure 22: Possible splitting points on a concave ice floe polygon and the eventually selected splitting crack (red).

Then, the shortest possible splitting crack is selected as the splitting propagation candidate: $min|\vec{r}^{start}_{crack} - \vec{r}^{end}_{crack}|$. The splitting crack is assumed to propagate if $R_{bending} < min|\vec{r}^{start}_{crack} - \vec{r}^{end}_{crack}| < S_k R_{bending}$, where $R_{bending}$ is the possible circumferential crack radius found according to Nevel (1961), and S_k is the splitting radius multiplication coefficient which can be adjusted in the software (Figure 23).

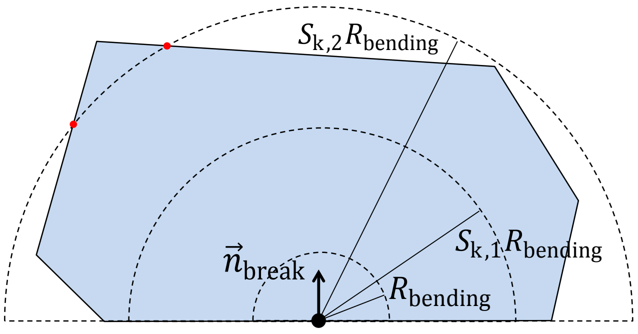

Figure 23: Splitting crack propagation criterion. Value of $S_{k,1}$ produces no splitting, while $S_{k,2}$ does lead to splitting.

If the splitting crack propagation is confirmed, two split floe polygons are created from the original ice floe (Figure 24). The first polygon contains all vertices from $\vec{r}^{start}_{crack}$ to $\vec{r}^{end}_{crack}$ in counter-clockwise order (blue and black vertices in Figure 24), while the second polygon contains all vertices from $\vec{r}^{end}_{crack}$ to $\vec{r}^{start}_{crack}$ in counter-clockwise order (black and red vertices in Figure 24).

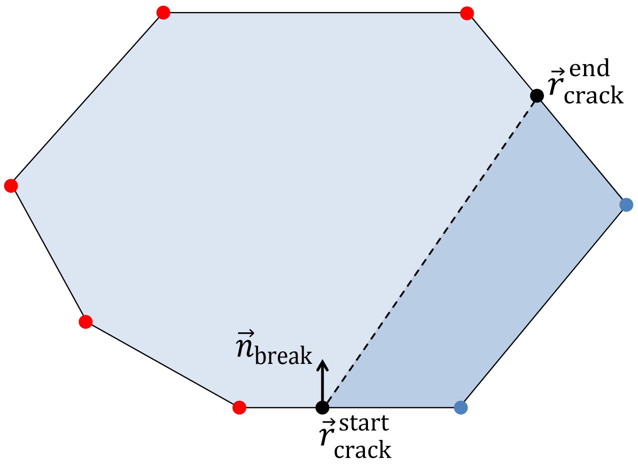
$\vec{r}^{\text{end}}_{\text{crack}}$
$\vec{n}_{\text{break}}$
$\vec{r}^{\text{start}}_{\text{crack}}$

Figure 24: Generation of the split floe polygons.

Then, these 2 polygons are checked for validity as follows. First, both polygons are checked for possible self-intersections by running Algorithm 8 on every edge of the polygon and checking that edge for intersections with all other edges constituting that same polygon. If any self-intersections are found, the polygon is rendered invalid and the splitting is not performed. The same happens if the area of any of the 2 polygons is below $2 \cdot 10^{-4}\ m^2$, because creation of tiny ice flakes is prohibited in the simulator for stability reasons. Finally, the object-aligned bounding boxes $\vec{b}_{box}$ of each polygon are found using Algorithm 9. If the ratios $b_{box,x}/b_{box,y}$ or $b_{box,y}/b_{box,x}$ are above R_{size} for any of the 2 polygons, the splitting is not performed. R_{size} is the user-defined floe narrowness parameter which is introduced to avoid creation of unphysically narrow floes in the simulator.

Algorithm 9 Object-aligned bounding box in 2D.

```
Amount of polygon vertices $N_{ver}$
Coordinates of polygon vertices $\vec{r}_i$
$j = N_{ver}$
$A_{min} = 10^{16} m^2$
for $i = 1 : N_{ver} - 1$ do
    $\vec{x}_{axis} = (\vec{r}_i - \vec{r}_j)/|\vec{r}_i - \vec{r}_j|$
    $\vec{y}_{axis} = (-x_{axis,y}, -x_{axis,x})^T$
    $min_x = min_y = max_x = max_y = 0$
    for $k = 1 : N_{ver}$ do
        $\vec{d} = \vec{r}_k - \vec{r}_j$
        $x_{proj} = \vec{d} \cdot \vec{x}_{axis}$
        $min_x \leftarrow min(min_x, x_{proj})$
        $max_x \leftarrow max(max_x, x_{proj})$
        $y_{proj} = \vec{d} \cdot \vec{y}_{axis}$
        $min_y \leftarrow min(min_y, y_{proj})$
        $max_y \leftarrow max(max_y, y_{proj})$
    end for
    $A = (max_x - min_x)(max_y - min_y)$
    if $A < A_{min}$ then
        $A_{min} = A$
        $\vec{b}_{box} = (max_x - min_x, max_y - min_y)^T$
    end if
    $j = i$
end for
Return $\vec{b}_{box}$
```

If the validity checks of the split floe polygons are successful, the splitting is finally performed and the fracture processing is finished. However, if the splitting is not performed, the bending failure can still take place as described in the following section.

5.5.3 Ice Bending Fracture

Bending fracture of a breakable ice floe is computed by the model of adjacent wedge-shaped beams resting on an elastic foundation (Lubbad and Løset, 2011). The circumferential crack is formed in the intersection patch if $\sigma_{wedge}^{max} \geq \sigma_f$, where σ_{wedge}^{max} is the maximal bending stress in the wedge-shaped ice beam resting on the elastic foundation, computed using the power series solution of Nevel (1961).

If the circumferential crack formation criterion is fulfilled, the polar angle of the first wedge crack θ_{crack} is found according to Algorithm 7. Then, a segmented circular polygon divided into $2N_{wedges}$ wedges is created by generating the following vertices in the 2D plane (Figure 25):

$$\begin{aligned}(r_{i,x}, r_{i,y})^T &= \\ &= (R_{bending} \cos i\alpha_{wedge}, R_{bending} \sin i\alpha_{wedge})^T\end{aligned}$$

where $i = 0 : 2N_{wedges} - 1$. Then, these vertices are transformed to the coordinates of the crack:

$$\begin{pmatrix} r_{i,x} \\ r_{i,y} \end{pmatrix} \leftarrow \begin{pmatrix} r_{i,x} \cos\theta_{crack} - r_{i,y} \sin\theta_{crack} + r_{crack,x}^{start} \\ r_{i,x} \sin\theta_{crack} + r_{i,y} \cos\theta_{crack} + r_{crack,y}^{start} \end{pmatrix}$$

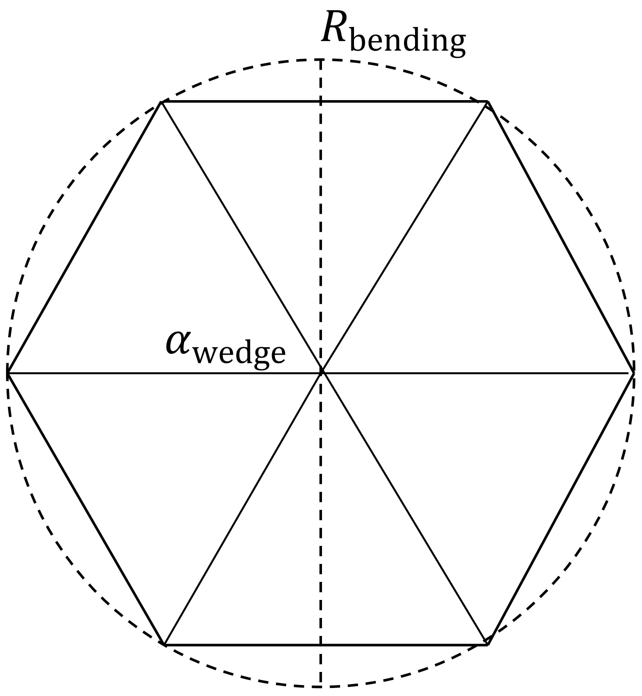

Figure 25: Segmentation of a circle into ice wedges. An example of $N_{wedges} = 3$ and $\alpha_{wedge} = \pi/3$.

Afterwards, this segmented polygon is subtracted from the original ice floe polygon using the *difference* function of the Boost.Geometry library (Figure 26). This procedure may produce several polygons out of the original one, as shown in Figure 27.

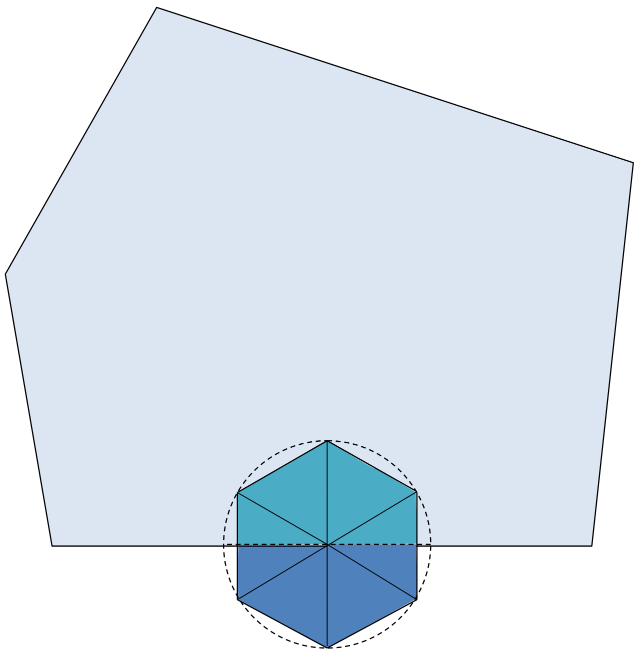

Figure 26: Bending failure polygon subtraction from a relatively large floe.

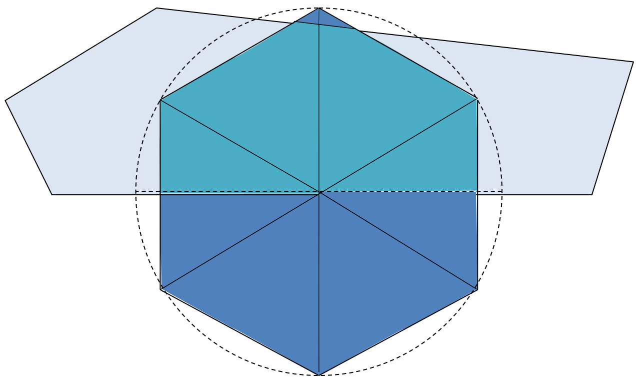

Figure 27: Bending failure polygon subtraction from a relatively small floe.

Next, the ice wedge triangles are created by generating the following 3 vertices:

$$(0,0)^T$$

$$(R_{bending}\cos i\alpha_{wedge}, R_{bending}\sin i\alpha_{wedge})^T$$

$$(R_{bending}\cos(i+1)\alpha_{wedge}, R_{bending}\sin(i+1)\alpha_{wedge})^T$$

where $i = 0 : 2N_{wedges} - 1$. Finally, these triangles are transformed to the coordinates of the crack using the above stated equations, and the intersections of each triangle with the original ice floe polygon are found using the *intersection* function of the Boost.Geometry library. The resultant intersection polygons are saved into the list of ice floe wedges for subsequent physical creation in the numerical environment. For example, in Figures 26 and 27, the light-blue polygons are the remnants of the original ice floe which will be re-created; the aquamarine polygons are the new ice wedges that will be created; and the dark-blue polygons are the areas of the bending failure stencil that are discarded by the intersection operation.

5.5.4 Generation of the Broken Ice Floes

After the initial polygons of the broken ice floes are created in Sections 5.5.2-5.5.3, the cleaning and simplification of their geometrical shapes are performed according to Algorithm 10. It removes polygonal edges shorter than 5 mm, deletes "spikes" and smoothens corners of the newly created ice floes (Figure 28). After the algorithm is finished, every processed polygon is checked for self-intersections. If they are found, it means that the cleaning algorithm has failed, and the original vertices of the polygon (i.e. pre-cleaning) are fully restored. Furthermore, if a processed polygon has less than 3 vertices (this could happen when cleaning very narrow triangles) or its area is below $2 \cdot 10^{-4}$ m^2, the polygon is just deleted from the simulation.

After the cleaning process is finished, the original breakable ice floe is fully deleted from the simulation through the PAL interface, and the new ice floes are introduced instead of it. To do that, every ice floe polygon is checked for convexity by ensuring that all cross products of consecutive edges have the same sign (Algorithm 11). If the polygon is convex, it is just extruded along the Z_g axis on the value of h_i in order to create the collision mesh of the new ice floe (as in Section 4.3). However, if the polygon is concave, it is first triangulated using constrained Delaunay triangulation library Poly2Tri (Green, 2013). During this process the software discards triangles with areas below 10^{-5} m^2 in order to avoid creation of degenerative collision meshes in the physical engine. Triangle areas for this check are computed as follows:

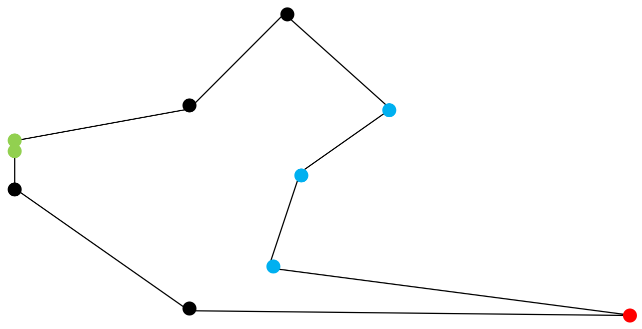

Figure 28: Polygon cleaning: the green vertices will be merged into 1, the red vertex is a "spike" which will be removed and the blue vertices form a smooth corner which will be transformed into a straight line by removing the middle vertex.

Algorithm 10 Polygon cleaning.

```
Amount of polygon vertices $N_{ver}$
Coordinates of polygon vertices $\vec{r}_i$
repeat
    Vertex removed $= false$
    $i = 0$
    while $i < N_{ver}$ do
        $\vec{e}_1 = \vec{r}_{i+1} - \vec{r}_i$
        $\vec{e}_0 = \vec{r}_i - \vec{r}_{i-1}$
        $\kappa = \frac{\vec{e}_1}{|\vec{e}_1|} \cdot \frac{\vec{e}_0}{|\vec{e}_0|}$
        if $|\vec{e}_1| < 0.005$ or $-0.99 > \kappa > 0.99$ then
            Vertex removed $= true$
            Delete $\vec{r}_i$
            $N_{ver} = N_{ver} - 1$
        else
            $i = i + 1$
        end if
    end while
until Vertex removed $= false$
```

$$A_{\triangle} = \frac{1}{2}[(r_{2,x}-r_{1,x})(r_{3,y}-r_{1,y})-(r_{3,x}-r_{1,x})(r_{2,y}-r_{1,y})]$$

where $\vec{r}_{1,2,3}$ are the 2D vertices of the triangle.

Algorithm 11 Polygon convexity check.

```
$\vec{e}_{N_{ver}} = \vec{r}_{N_{ver}} - \vec{r}_{N_{ver}-1}$
$\vec{e}_1 = \vec{r}_1 - \vec{r}_{N_{ver}}$
$\vec{e}_2 = \vec{r}_2 - \vec{r}_1$
$s_{N_{ver}} = e_{N_{ver},x}e_{1,y} - e_{N_{ver},y}e_{1,x}$
$s_1 = e_{1,x}e_{2,y} - e_{1,y}e_{2,x}$
if $s_1 \cdot s_{N_{ver}} < 0$ then
    Polygon is concave, exit algorithm
end if
$s_{prev} = s_1$
$\vec{e}_{prev} = \vec{e}_1$
for $i = 2 : N_{ver} - 2$ do
    $\vec{e} = \vec{r}_{i+1} - \vec{r}_i$
    $s = e_{prev,x}e_y - e_{prev,y}e_x$
    if $s_{prev} \cdot s < 0$ then
        Polygon is concave, exit algorithm
    end if
    $s_{prev} = s$
    $\vec{e}_{prev} = \vec{e}$
end for
Polygon is convex
```

Then, each triangle of the concave ice floe is extruded into a 3D prism of h_i thickness, and the collision mesh of the floe is constructed as a compound collection of these triangular prisms, i.e. the final collision mesh is the same as shown in Figure 9, c. It means that the side faces of the collision mesh are not triangulated, because it is needed only for the fluid mesh. As for the actual fluid meshes of the new ice floes, they are constructed using the same method and source code as presented in Section 4.3.

The remainder of the fractured level ice sheet remains a static body in the physical engine, while the new broken ice pieces are introduced as dynamic bodies with the same thicknesses and densities as the parent ice sheet. Their initial linear velocities are computed as follows:

$$\vec{v}_{new} = \vec{v}_{old} + \vec{\Omega} \times (\vec{r}^{\,new}_{CoM} - \vec{r}^{\,old}_{CoM})$$

where $\vec{v}_{old}$ and $\vec{\Omega}$ are the linear and angular velocities of the original ice floe before fracture (but after the multibody solving), $\vec{r}^{\,new}_{CoM}$ is the center of mass of the newly broken ice floe in the global coordinate system and $\vec{r}^{\,old}_{CoM}$ is the center of mass of the original floe before breakage in the global coordinate system (i.e. the velocity pivot point).

5.6 Time Integration

The final step of the simulation loop is the time integration which updates the positions of all bodies and joints in the simulation domain. The following fixed-time-step integration scheme is implemented in the software:

$$\vec{r}_{new} = \vec{r}_{old} + \triangle t H \vec{u}_{new}$$

where $\vec{r} = (\vec{r}_1^T, Q_1^T, \vec{r}_2^T, Q_2^T, \cdots, \vec{r}_{N_{body}}^T, Q_{N_{body}}^T)^T \in \mathbb{R}^{7N_{body}\times 1}$ is the generalized position vector of the system and

$$H = \begin{pmatrix} I_{3\times 3} & 0 & 0 & 0 & \cdots & 0 \\ 0 & G_1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & I_{3\times 3} & 0 & \cdots & 0 \\ 0 & 0 & 0 & G_2 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & I_{3\times 3} & 0 \\ 0 & 0 & 0 & \cdots & 0 & G_{N_{body}} \end{pmatrix} \in \mathbb{R}^{7N_{body}\times 6N_{body}}$$

is the kinematic map of the multibody system, where

$$G_i = 0.5 \begin{pmatrix} -Q_x^i & -Q_y^i & -Q_z^i \\ Q_w^i & Q_z^i & -Q_y^i \\ -Q_z^i & Q_w^i & Q_x^i \\ Q_y^i & -Q_x^i & Q_w^i \end{pmatrix} \in \mathbb{R}^{4\times 3}$$

is the transformation matrix between the angular velocity and the quaternion derivative of body i. $\vec{u}_{new} \in \mathbb{R}^{6N_{body}\times 1}$ is the updated system velocity vector from Section 5.4.

Although the updated positions are supposed to resolve any penetrations between the bodies, due to discrete nature of the collision detection system some unresolved penetrations may remain after the position update step. These penetrations are corrected using the "split impulse" position correction method of the Bullet engine. This method first solves the following LCP using the same Projected Gauss-Seidel method as in Section 5.4:

$$J_{cont} M^{-1} J_{cont}^T \vec{p}^*_{cont} - \vec{b}_{pen} \geq 0 \perp 0 \leq \vec{p}^*_{cont} \leq +\infty$$

where $\vec{p}^*_{cont} \in \mathbb{R}^{3N_{cont}\times 1}$ is the "pseudo" contact impulse vector of the system and $\vec{b}_{pen} = -\frac{\beta}{\triangle t}(d^1_{pen}, 0, 0, d^2_{pen}, 0, 0, \cdots, d^{N_{cont}}_{pen}, 0, 0)^T \in \mathbb{R}^{3N_{cont}\times 1}$ is the position correction vector of the system. d^i_{pen} is the penetration depth in contact i reported by the narrow phase collision detection system, $\beta = min(\beta_0, 1 - \vec{p}^*_{cont,n}/p_{lim})$ is the error reduction parameter, $\beta_0 = 0.8$ and p_{lim} is the ice crushing constraint value introduced in Section 5.2.2.

Then, the "pseudo" velocity vector of the system is computed as follows:

$$\vec{u}^* = M^{-1} \triangle t J_{cont}^T \frac{\vec{p}^*_{cont}}{\triangle t} \in \mathbb{R}^{6N_{body}\times 1}$$

Finally, the position vector of the system is updated using the "pseudo" velocity:

$$\vec{r}_{new} = \vec{r}_{old} + \triangle t H \vec{u}^*$$

This means that moving objects out of penetrations does not add any extra kinetic energy to the system. An alternative to this method is the Baumgarte position correction scheme (Catto, 2005) enabled in the Bullet engine by default. However, it was found to generate unphysically high forces during ice breaking simulations, which were leading to "explosions" in the ice field. Therefore, it was decided to use the "split impulse" method instead. However, the position correction is disabled for the contacts between the breakable ice floes and the vessel, in order to avoid interference with the ice breaking force computations and ensure correct contact response of the vessel. Technically, modification of the error reduction parameter and disabling of the position correction for a subset of contact constraints is implemented as a functional callback to the time integration step of the Bullet engine.

6 Simulator Output and Post-processing

This section describes the output data of the software tool. In addition to numerical (Section 6.1) and visualization (Section 6.2) output data channels, the software maintains an open console window throughout the whole simulation process. This window displays the simulation progress towards completion (in percent), which is updated at every time step. Moreover, the console window displays the logging messages generated by the simulator, such as exception messages or any information messages coming from the underlying physical engine. This logging process is managed by the PAL library. Finally, the console window displays to the user the total wall-clock time used by a simulation at the end of the whole simulation process.

6.1 Numerical Data

At every time step the simulator writes into the output CSV file the floating-point values, in fixed-point notation, of the following quantities (all of them in the $\{X_g, Y_g, Z_g\}$ frame): contact loads, carriage joint loads, thruster loads, position and orientation of the structure and its linear and angular velocities. All these data can then be post-processed using a data analysis package, e.g. Matlab.

The contact load channel contains the forces exerted on the vessel by the ice floes, which are calculated by adding together the contact forces and torques from every contact point between the structure and the ice floes (both breakable and unbreakable). The contact force in a single contact point is computed as follows: $\vec{f_c} = (p_n\vec{n} + f_\tau^1\vec{\tau}_1 + f_\tau^2\vec{\tau}_2)/\triangle t$, where p_n is the normal constraint impulse computed by the multibody solver of the physical engine, $\vec{n}$ is the contact normal from the narrow phase collision detection system, f_τ^1, f_τ^2 are the frictional constraint impulses in 2 orthogonal directions $\vec{\tau}_1$ and $\vec{\tau}_2$, and $\triangle t$ is the time step. The corresponding contact torques on the structure and the ice floe are: $\vec{\tau}_c^1 = (\vec{r}_c^1 - \vec{r}_{CoM}^1) \times \vec{f_c}$ and $\vec{\tau}_c^2 = -(\vec{r}_c^2 - \vec{r}_{CoM}^2) \times \vec{f_c}$, where $\vec{r}_c^1$ and $\vec{r}_c^2$ are the witness points on the vessel and the contacting ice floe, and $\vec{r}_{CoM}^1$ and $\vec{r}_{CoM}^2$ are the center of mass positions of the contacting bodies.

The carriage joint load channel contains the reaction force and torque applied to the structure by the carriage joint: $\vec{f}_{joint}$ and $\vec{\tau}_{joint}$. They are computed by the multibody solver (Section 5.4). The CSV channel contains $\vec{f}_{joint}$ for the force and $\vec{\tau}_{joint} + (\vec{r}_{CoM} - \vec{r}_{joint}) \times \vec{f}_{joint}$ for the torque, where $\vec{r}_{CoM}$ is the center of mass position of the vessel and $\vec{r}_{joint}$ is the position of the joint. The hull loads CSV channel contains the same force values, but the torque is calculated relative to the center of mass of the structure, i.e. the pure value of $\vec{\tau}_{joint}$ is reported to the user.

The thruster loads channel contains the forces and torques produced by the propellers of the vessel in free running or DP simulation modes.

The position of the structure is reported into the corresponding CSV channel directly from $\vec{r}_{CoM}$, while its orientation yaw-pitch-roll angles are computed from the orientation quaternion as follows:

$$\begin{aligned}
y &= atan2\frac{2(Q_xQ_y + Q_wQ_z)}{Q_w^2 + Q_x^2 - Q_y^2 - Q_z^2}\\
p &= \sin^{-1} 2(Q_wQ_y - Q_xQ_z)\\
r &= atan2\frac{2(Q_yQ_z + Q_wQ_x)}{Q_w^2 - Q_x^2 - Q_y^2 + Q_z^2}
\end{aligned}$$

The linear velocity of the vessel is reported directly from $\vec{v}_{CoM}$ into the output CSV file, while the angular velocity is converted into roll-pitch-yaw rates using the method of Ardakani and Bridges (2010) as follows. First, the inverse unit-norm quaternion of the vessel is computed:

$$Q^{-1} = \frac{(-Q_x, -Q_y, -Q_z, Q_w)^T}{Q_x^2 + Q_y^2 + Q_z^2 + Q_w^2}$$

Then, the quaternion-vector multiplication is performed to rotate the angular velocity vector of the vessel:

$$\begin{aligned}
&Q^{-1}\vec{\Omega} =\\
&= (Q_w^{-1}\Omega_x + Q_y^{-1}\Omega_z - Q_z^{-1}\Omega_y,\\
&Q_w^{-1}\Omega_y + Q_z^{-1}\Omega_x - Q_x^{-1}\Omega_z,\\
&Q_w^{-1}\Omega_z + Q_x^{-1}\Omega_y - Q_y^{-1}\Omega_x,\\
&-Q_x^{-1}\Omega_x - Q_y^{-1}\Omega_y - Q_z^{-1}\Omega_z)^T
\end{aligned}$$

Next, this value is used to obtain the angular velocity of the vessel in the local frame $\{X_s, Y_s, Z_s\}$:

$$\begin{aligned}
&\vec{\Omega}_{loc} =\\
&(Q_x(Q^{-1}\vec{\Omega})_w + Q_w(Q^{-1}\vec{\Omega})_x + Q_z(Q^{-1}\vec{\Omega})_y - Q_y(Q^{-1}\vec{\Omega})_z\\
&Q_y(Q^{-1}\vec{\Omega})_w + Q_w(Q^{-1}\vec{\Omega})_y + Q_z(Q^{-1}\vec{\Omega})_x - Q_z(Q^{-1}\vec{\Omega})_x\\
&Q_z(Q^{-1}\vec{\Omega})_w + Q_w(Q^{-1}\vec{\Omega})_z + Q_x(Q^{-1}\vec{\Omega})_y - Q_x(Q^{-1}\vec{\Omega})_y)^T
\end{aligned}$$

Finally, the roll-pitch-yaw rates of the vessel are computed as follows:

$$\begin{aligned}
\dot{r} &= \Omega_{loc,x} + \frac{\Omega_{loc,y}\sin r \sin p}{\cos p} + \frac{\Omega_{loc,z}\cos r \sin p}{\cos p}\\
\dot{p} &= \Omega_{loc,y}\cos r - \Omega_{loc,z}\sin r\\
\dot{y} &= \frac{\Omega_{loc,y}\sin r}{\cos p} + \frac{\Omega_{loc,z}\cos r}{\cos p}
\end{aligned}$$

This algorithm fails if $\cos p = 0$, because of the Gimbal lock condition (when it is impossible to convert the angular velocity into the roll-pitch-yaw rates). Then, zero values are reported into the output CSV channel.

6.2 Visualization

The software utilizes the Irrlicht library version 1.7.3 (Gebhardt et al., 2012) for visualizing the numerical scene. Every simulated object (the vessel, the ice floes, the fluid domain, the walls of the ice tank and the towing carriage) gets associated with a corresponding "scene node" of the Irrlicht engine for this purpose. The ice tank walls, the towing carriage and the fluid domain are visualized as rectangular cuboids with a certain color and transparency, while the vessel and the ice floes are visualized by either their fluid or collision meshes, depending on the user's choice (Figure 29, a-b). Both mesh types can be visualized either in solid or wireframe modes (Figure 29, c). Collision meshes of the objects are visualized with random colors (Figure 29, b), while fluid meshes are always visualized with white color (Figure 29, a). Finally, the contact normals reported by the narrow phase collision detection system of the physical engine can be visualized by red patches connecting the witness points between the colliding bodies (Figure 29, d).

At every time step the positions and orientations of all dynamic bodies from the physical engine are synchronized with their respective Irrlicht scene nodes to advance the visualization in real time. However, the ice tank walls and the fluid domain are static and do not need to be updated. After the synchronization, all scene nodes are redrawn by the Irrlicht engine at every timestep. If an ice floe gets broken, its scene node is deleted and scene nodes of the newly broken ice pieces are created instead of it.

16bit OpenGL rendering configuration with the user's screen resolution is used by the software for the visualization window. The colors of the ambient environment and the background can be adjusted in the source code of the software. Furthermore, additional light sources can be added to illuminate certain parts of the simulated scene. By default, two light sources are created: one above and one below the ice sheet.

A camera is added to the visualization window at a certain elevation, heading and distance to the target point, as specified by the user in the input XML file. It is possible to rotate, pan and zoom the camera using the mouse (as implemented through the *ISceneNodeAnimator* class of the Irrlicht engine). The motion speed of the camera is automatically adjusted by the software according to the scale of the simulated scene.

Finally, an information palette is displayed in the top-left corner of the visualization window (Figure 29, a-d). It contains an information message on how to control the camera and how to enable/disable the wireframe and collision geometry views in the simulated scene. Furthermore, this window includes the timing information of the various software modules. Namely, the fluid force calculation time, the physics engine step time (including ice force calculation) and the geometrical ice breaking time (old floe deletion and new floe creation) are measured at every time step and displayed to the user in [ms] for the purpose of rapid profiling. All timings in the software are measured with the precision of CPU frequency.

The user has a possibility of pausing the visualization at any time to inspect the simulated scene. Otherwise, the simulation will continue until the user either closes the visualization window or the total amount of simulation steps is reached.

7 Discussion

Although the presented software framework is a powerful tool for simulating vessels and offshore structures in ice, it has several major limitations stemming from the simplifying assumptions of the computational methods used by the model. These limitations are discussed in this Section of the paper.

The first major assumption of the numerical model is that both the ice floes and the vessel are treated as rigid bodies in the simulation. It means that local deformations of the objects, which arise in contact regions, are not simulated. Although this could be an acceptable assumption for the vessel, local plastic deformations of the ice floes (e.g. crushing and creep) could be very important to capture for estimating global ice actions in certain environmental conditions (such as, for example, in late autumn ice). Furthermore, local material extrusion due to ice crushing and micro cracks in the ice (material damage) are not modelled in the current version of the simulator (they are bypassed with a limiting value in the ice crushing constraint, as described in Section 5.2.2). Although this approach could be acceptable for sloping-sided structures, its validity for vertically-sided structures could be questionable. Therefore, additional research is suggested on the topic of local ice deformations at the contact interface with vertically-sided floating structures.

The fluid force model described in Section 5.3.1 has several limitations: there are no free surface effects, no volumetric deformations of the fluid and no interactions of the objects through the fluid domain. Moreover, only buoyancy and drag forces are simulated, while the added mass and damping effects are not captured. Although this model has produced reasonable simulation results at low ice-structure interaction velocities, improvements of the hydrodynamic formulations and validation of the current drag model may contribute to increased validity range of the simulation tool.

Inertia tensor and center of mass of the vessel are currently generated by the application assuming uniform mass distribution inside the fluid mesh. The user does not have a possibility to assign a pre-defined inertia tensor or change the position of the center of gravity of the vessel. This aspect of the simulation tool should be improved if more extensive engineering use is envisaged for the application. Furthermore, the meshing procedures of the pre-processor could be improved. Namely, the automation of the convex decomposition process and support of non-triangular faces in the collision mesh of the vessel would increase the usage efficiency of the software tool.

In the current version of the model the towing carriage joint is implemented as a single prismatic link between the structure and the carriage. In real model testing the joint consists of 6 load cells constituting the force balance system. This discrepancy might lead to

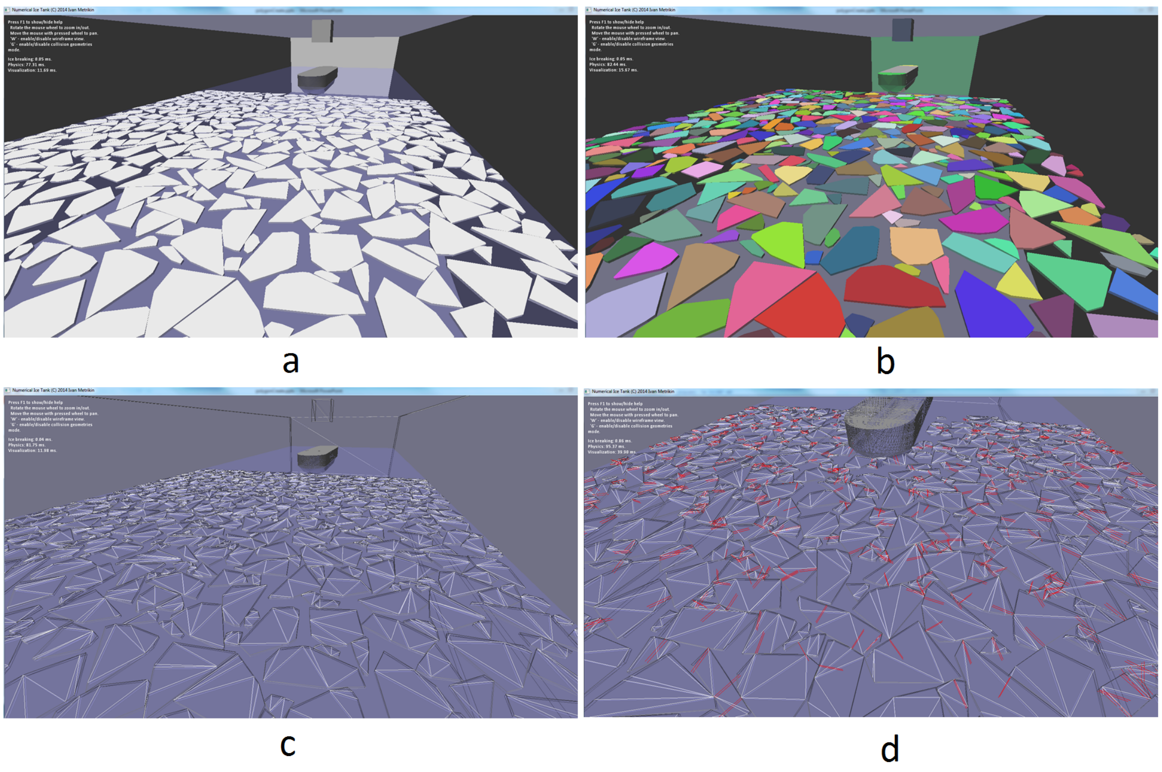

Figure 29: Visualization functionalities of the software framework. a - fluid mesh view, solid. b - collision mesh view, solid. c - fluid mesh view, wireframe. d - contact normals in wireframe view.

imprecise simulation of the towing process and, therefore, requires further attention when simulating oblique towing tests.

The broken ice field generation algorithm described in Section 4.3 suffers from a major deficiency: it takes a very long time to produce ice concentrations above 80%. Additional research in this field would, therefore, be beneficial to increase the efficiency and applicability range of the ice field generation module.

Currently all ice floes in the ice field have uniform and constant properties (thickness, density, strength). In reality those properties are statistically distributed throughout the ice sheet and even within individual ice floes (including those in a model basin). Therefore, implementation of a statistical routine to generate ice property distributions could benefit the numerical tool.

Geometrical algorithms that compute intersections between breakable ice floes and the vessel are based on a simplified 2D treatment of the contact problem. Furthermore, the fluid mesh of the structure is used in those algorithms for computing the possible ice fracture direction vectors. It could be advantageous to use the native collision detection system of the physical engine to formulate the ice crushing constraint, because it would leave out the need to maintain 2 different collision systems in 1 application. Moreover, it would be more natural to use the collision mesh of the vessel for the ice breaking algorithms instead of the fluid mesh. However, the current collision detection system used by the physical engines is discrete, i.e. the simulation suffers from unavoidable penetrations between the objects. Implementation of a continuous collision detection routine may improve the overall simulation accuracy and stability. Finally, line segments shorter than 10^{-6} m are currently not allowed in the simulator to avoid geometrical routine degeneracies. As a consequence, very small length scales cannot be handled correctly by the model. Therefore, additional research on the above mentioned geometrical and collision detection aspects of the software tool is suggested.

In the current implementation of the ice fracture routines it is assumed that cracks in the ice propagate instantly and along straight lines without any curvature. In reality, however, there can be curved cracks going along the ridges embedded into ice floes. Moreover, the ice splitting is currently treated as a geometrical problem rather than physical problem, which limits the realism of the simulation tool. Recently, a large body of novel research work related to ice fracture initiation, fracture propagation and breakability criteria has been reported by Lu (2014). It is expected that incorporation of these new results into the simulator would substantially improve the realism and physical correctness of the ice breaking module in the numerical tool.

Finally, the multibody solver described in Section 5.4 uses a 2-directional friction pyramid instead of the full Coulomb cone (see Section 4.6 in Erleben (2005)). This simplification leads to anisotropy of the frictional forces and may substantially decrease the validity of simulation results. Furthermore, the static friction coefficient is assumed to be always equal to dynamic friction coefficient, which is not a fully realistic assumption of the ice material. Moreover, the Bullet physical engine disables the gyroscopic torque term for all bodies in the simulation domain, which may lead to even further decreased accuracy of the simulation results. A better multibody model, such as the one of Tasora and Anitescu (2011), could improve this aspect of the simulator. Furthermore, a rolling friction model might be needed to model dense ice accumulations and rubble fields. The method of Tasora and Anitescu (2013) seems very promising for addressing this challenge. Furthermore, the currently utilized Projected Gauss-Seidel iterative multibody solver does not guarantee convergence of the solving loop with a fixed amount of iterations. A better LCP solving method could improve simulation results substantially, for example the one described in Heyn et al. (2013). Alternatively, an error control routine of the Projected Gauss-Seidel loop could be implemented in the future version of the simulator. Finally, the split impulse position correction method of the Bullet engine perturbs the energy of the physical system and may lead to incorrect simulation results. A better stabilization method should be implemented, e.g. the one of Tasora and Anitescu (2011), in order to improve the realism of the simulated contact response.

8 Conclusions

This paper presents a software package developed by the Norwegian University of Science and Technology for simulating stationkeeping operations of vessels and offshore structures in discontinuous floating sea ice. The package has been successfully used and cross-checked against experimental data in various academic and industrial projects: Metrikin et al. (2013b); Kerkeni et al. (2013b,a); Kerkeni and Metrikin (2013); Metrikin and Løset (2013); Metrikin et al. (2013a); Kerkeni et al. (2014); Scibilia et al. (2014); Østhus (2014); Kjerstad and Skjetne (2014).

Both theoretical and software implementation aspects of the numerical model have been elucidated in the paper:

- Input file structure
- Mesh preparation process
- Creation of the vessel and the towing carriage in the physical engine
- Ice field generation
- Collision detection
- Ice fracture
- Fluid force computation
- Multibody solving process
- Position correction and time stepping routines

Finally, several limitations of the computational techniques have been exposed and discussed. The most important ones, from the author’s perspective, are:

- Collision detection system of the physical engine
- Iterative multibody solver of the physical engine
- Split impulse position correction method of the physical engine
- Anisotropic and simplified friction model of the physical engine
- Simplified treatment of the ice fracture problem
- Absence of local material extrusion due to ice crushing
- Crude hydrodynamic model

Acknowledgments

Research partly funded by the Research Council of Norway (RCN) project no. 196897: ”MARTEC ERA-NET DYPIC: Dynamic positioning in ice covered waters”, supported by the Federal Ministry of Economics and Technology (Germany) and Ministry of Ecology, Sustainable Development, Transport and Housing (France) with partners HSVA, Sirehna, Statoil, DNV GL, Kongsberg Maritime and NTNU and partly by KMB project no. 199567: Arctic DP, with partners Kongsberg Maritime, Statoil and DNV GL.

The author is grateful to Professor Sveinung Løset for inspiring creation of the simulator, initiating the work, securing resources for the project, providing valuable advice throughout the development process and securing experimental data for validating and calibrating the software tool. Furthermore, the author would like to thank Mr. Dmitry Sapelnikov for developing the ice field generation algorithms and implementing the software framework, Mr. Øivind K. Kjerstad and Mr. Sofien Kerkeni for uncovering and reporting issues in the numerical model and for valuable technical discussions. Ms. Capucine Thomas-Lépine is thanked for making the initial technical drawings for Section 5.5 of the paper. Dr. ing. Nils Albert Jenssen and Mr. Quentin Hisette are thanked for reviewing the first version of the manuscript and providing constructive feedback that contributed significantly to improving the quality of the paper.

Finally, Statoil ASA is gratefully acknowledged for funding the final stages of research and permitting the author to write and publish this paper.

References

Algoryx Simulation AB. AgX Dynamics library. https://www.algoryx.se/products/agx-dynamics/, 2014. [Accessed 1 Dec 2014].

Ardakani, H. A. and Bridges, T. J. Review of the 3-2-1 Euler Angles: a yaw-pitch-roll sequence. Technical report, Department of Mathematics, University of Surrey, 2010.

Autodesk. 3ds Max. http://www.autodesk.com/products/3ds-max/, 2014. [Accessed 1 Dec 2014].

Blender Foundation. Blender. http://www.blender.org/, 2014. [Accessed 1 Dec 2014].

Boeing, A. Physics Abstraction Layer library. http://www.adrianboeing.com/pal/, 2009. [Accessed 1 Dec 2014].

Bonnemaire, B., Lundamo, T., Serré, N., and Fredriksen, A. Numerical Simulations of Moored Structures in Ice: Influence of Varying Ice Parameters. In *Proc. of the 21st Intl. Conf. on Port and Ocean Engineering under Arctic Conditions (POAC'11)*. 2011.

Catto, E. Iterative Dynamics with Temporal Coherence. Technical report, Crystal Dynamics, 2005.

Catto, E. Exact Buoyancy for Polyhedra. In M. Dickheiser, editor, *Game Programming Gems 6*, pages 175–188. Charles River Media, 2006.

CM Labs Simulations Inc. Vortex Dynamics. http://www.cm-labs.com/market/robotics/products/vortex-dynamics-software/, 2014. [Accessed 1 Dec 2014].

Coumans, E. Bullet Physics library, version 2.81. http://bulletphysics.org/, 2012. [Accessed 1 Dec 2014].

Coumans, E. Physics for Game Programmers: Exploring MLCP and Featherstone Solvers. In *Proc. of the Game Developers Conference*. 2014.

Dawes, B. Boost.Filesystem library. http://www.boost.org/doc/libs/1_57_0/libs/filesystem/doc/, 2012. [Accessed 1 Dec 2014].

Derradji-Aouat, A. Critical roles of constitutive laws and numerical models in the design and development of arctic offshore installations. In *Proc. of the International Conference and Exhibition on Ships and Structures in Ice (ICETECH)*. Banff, Alberta, Canada, 2010.

Erleben, K. *Stable, Robust, and Versatile Multibody Dynamics Animation*. Phd thesis, University of Copenhagen, 2005.

Gebhardt et al. Irrlicht Engine. http://irrlicht.sourceforge.net/, 2012. [Accessed 1 Dec 2014].

Gehrels, B., Lalande, B., Loskot, M., and Wulkiewicz, A. Boost.Geometry library. http://www.boost.org/doc/libs/1_57_0/libs/geometry/doc/html/index.html, 2014. [Accessed 1 Dec 2014].

Green, M. Constrained Delaunay triangulation library Poly2Tri. http://code.google.com/p/poly2tri/, 2013. [Accessed 1 Dec 2014].

Hamilton, J. M. The Challenges of Deep-Water Arctic Development. *International Journal of Offshore and Polar Engineering*, 2011. 21(4).

Heyn, T., Anitescu, M., Tasora, A., and Negrut, D. Using Krylov subspace and spectral methods for solving complementarity problems in many-body contact dynamics simulation. *International Journal for Numerical Methods in Engineering*, 2013. 95(7):541–561. doi:10.1002/nme.4513.

Kapoulkine, A. Pugixml library. http://pugixml.org/, 2014. [Accessed 1 Dec 2014].

Kerkeni, S., Dal Santo, X., Doucy, O., Jochmann, P., Haase, A., Metrikin, I., Løset, S., Jenssen, N. A., Hals, T., Gürtner, A., Moslet, P. O., and Støle-Hentschel, S. DYPIC Project: Technological and Scientific Progress Opening New Perspectives. In *Proc. of Arctic Technology Conference*. 2014. doi:10.4043/24652-MS.

Kerkeni, S., Dal Santo, X., and Metrikin, I. Dynamic positioning in ice: Comparison of control laws in open water and ice. In *Proc. of ASME 2013 32nd International Conference on Ocean, Offshore and*

Arctic Engineering. 2013a. doi:10.1115/OMAE2013-10918.

Kerkeni, S. and Metrikin, I. Automatic Heading Control for Dynamic Positioning in Ice. In *Proc. of MTS Dynamic Positioning Conference*. 2013.

Kerkeni, S., Metrikin, I., and Jochmann, P. Capability plots of dynamic positioning in ice. In *Proc. of ASME 2013 32nd International Conference on Ocean, Offshore and Arctic Engineering*. 2013b. doi:10.1115/OMAE2013-10912.

Kjerstad, Ø. K. and Skjetne, R. Modeling and Control for Dynamic Positioned Marine Vessels in Drifting Managed Sea Ice. *Modeling, Identification and Control*, 2014. 35(4):249–262. doi:10.4173/mic.2014.4.3.

Lee, S.-G., Zhao, T., Kim, G.-S., and Park, K.-D. Ice resistance test simulation of arctic cargo vessel using FSI analysis technique. In *Proc. of the International Offshore and Polar Engineering Conference*. pages 1162–1168, 2013.

Lindenhof, W. Wavefront OBJ Mesh Loader library. `http://www.limegarden.net/2010/03/02/wavefront-obj-mesh-loader/`, 2012. [Accessed 1 Dec 2014].

Lu, W. *Floe Ice - Sloping Structure Interactions*. Phd thesis, Norwegian University of Science and Technology, 2014.

Lubbad, R. and Løset, S. A numerical model for real-time simulation of ship-ice interaction. *Cold Regions Science and Technology*, 2011. 65(2):111–127. doi:10.1016/j.coldregions.2010.09.004.

Mamou, K. Hierarchical Approximate Convex Decomposition library. `http://hacd.sourceforge.net/`, 2013. [Accessed 1 Dec 2014].

Metrikin, I., Borzov, A., Lubbad, R., and Løset, S. Numerical Simulation of a Floater in a Broken-Ice Field. Part II: Comparative Study of Physics Engines. In *Proc. of ASME 2012 31st International Conference on Ocean, Offshore and Arctic Engineering (OMAE2012)*. 2012a. doi:10.1115/OMAE2012-83430.

Metrikin, I., Kerkeni, S., Jochmann, P., and Løset, S. Experimental and Numerical Investigation of Dynamic Positioning in Level Ice. In *Proc. of ASME 2013 32nd International Conference on Ocean, Offshore and Arctic Engineering (OMAE2013)*. 2013a. doi:10.1115/OMAE2013-10910.

Metrikin, I. and Løset, S. Nonsmooth 3D Discrete Element Simulation of a Drillship in Discontinuous Ice. In *Proc. of 22nd International Conference on Port and Ocean Engineering under Arctic Conditions (POAC'13)*. 2013.

Metrikin, I., Løset, S., Jenssen, N. A., and Kerkeni, S. Numerical Simulation of Dynamic Positioning in Ice. *Marine Technology Society Journal*, 2013b. 47(2):14–30. doi:10.4031/MTSJ.47.2.2.

Metrikin, I., Lu, W., Lubbad, R., Løset, S., and Kashafutdinov, M. Numerical Simulation of a Floater in a Broken-Ice Field. Part I: Model Description. In *Proc. of ASME 2012 31st International Conference on Ocean, Offshore and Arctic Engineering (OMAE2012)*. 2012b. doi:10.1115/OMAE2012-83938.

Nevel, D. E. The Narrow Free Infinite Wedge on an Elastic Foundation. Technical report, 1961.

Nevel, D. E. Ice forces on cones from floes. In *Proc. of IAHR Ice Symposium*. pages 1391–1404, 1992.

Nvidia Corporation. NVIDIA PhysX library. `https://developer.nvidia.com//gameworks-physx-overview/`, 2014. [Accessed 1 Dec 2014].

Østhus, V. *Robust Adaptive Control of a Surface Vessel in Managed Ice Using Hybrid Position- and Force Control*. Msc thesis, Norwegian University of Science and Technology, 2014.

Palmer, A. C. and Croasdale, K. R. *Arctic Offshore Engineering*. World Scientific Publishing Co. Pte. Ltd., 2013.

Scibilia, F., Metrikin, I., Gürtner, A., and Teigen, S. H. Full-scale trials and numerical modeling of sea ice management in the greenland sea. In *Proc. of the Arctic Technology Conference*. Houston, Texas, USA, 2014. doi:10.4043/24643-MS.

Servin, M., Wang, D., Lacoursire, C., and Bodin, K. Examining the smooth and nonsmooth discrete element approaches to granular matter. *International Journal for Numerical Methods in Engineering*, 2014. 97(12):878–902. doi:10.1002/nme.4612.

Skjetne, R. Arctic DP: The Arctic Offshore Project on Stationkeeping in Ice. In *Proc. of the IBC Maritime Ice Class Vessels Conference*. 2014. doi:10.13140/2.1.2843.1360.

Smith, R. Open Dynamics Engine library. `http://www.ode.org/`, 2014. [Accessed 1 Dec 2014].

Su, B. *Numerical Predictions of Global and Local Ice Loads on Ships*. Phd thesis, Norwegian University of Science and Technology, 2011.

Tasora, A. and Anitescu, M. A matrix-free cone complementarity approach for solving large-scale, nonsmooth, rigid body dynamics. *Computer Methods in Applied Mechanics and Engineering*, 2011. 200(5-8):439–453. doi:10.1016/j.cma.2010.06.030.

Tasora, A. and Anitescu, M. A complementarity-based rolling friction model for rigid contacts. *Meccanica*, 2013. 48(7):1643–1659. doi:10.1007/s11012-013-9694-y.

Tonon, F. Explicit Exact Formulas for the 3-D Tetrahedron Inertia Tensor in Terms of its Vertex Coordinates. *Journal of Mathematics and Statistics*, 2005. 1(1):8–11. doi:10.3844/jmssp.2005.8.11.

Valanto, P. and Puntigliano, F. M. E. A numerial prediction of the icebreaking resistance of a ship in level ice: Final report. Technical report, Hamburg Ship Model Basin, 1997.

Wang, J. and Derradji-Aouat, A. Ship Performance in Broken Ice Floes - Preliminary Numerical Simulations. Technical report, National Research Council of Canada, Institute for Ocean Technology, 2010.

Wang, J. and Derradji-Aouat, A. Numerical Assessment for Stationary Structure (Kulluk) in Moving Broken Ice. In *Proc. of the 21st International Conference on Port and Ocean Engineering under Arctic Conditions*. 2011.

Wright, B. Evaluation of Full Scale Data for Moored Vessel Stationkeeping in Pack Ice (With Reference to Grand Banks Development). Technical report, B. Wright and Associates Ltd., 1999.