Siku

Sea Ice Discrete Element Method Model
Final Report and Documentation

Authors:
Anton V Kulchitsky
Jennifer Hutchings
Jerome B Johnson

June 04, 2013 – August 16, 2015

branch: master
version: 0.1.0

This study is sponsored by the University of Alaska Coastal Marine Institute. Study collaboration and funding were provided by the U.S. Department of the Interior, Bureau of Ocean Energy Management, Environmental Studies Program under Agreement Number M13AC00001.

The matching funds are provided by Institute of Northern Engineering, University of Alaska Fairbanks and by Oregon State University.

This work was supported in part by a grant of HPC resources from the Arctic Region Supercomputing Center at the University of Alaska Fairbanks.

Contents

Chapter 1

Introduction

1.1 Basic Information

Siku is written on a subset of C++ language. Please, see Sec. 1.3 for strict programming policy.

1.2 Software Requirements

To compile and use the model, the following packages are required:

1. Python 3.0 or higher
   (a) `mathutils` from blender project. See `https://code.google.com/p/blender-mathutils/`.
   (b) `netcdf4-python` (recommended if you need import NetCDF4 forcing files).
   (c) `numpy`
   (d) `shapefile` (known also as `pyshp`).
2. HDF5 version 1.8 or higher.
3. NetCDF4 version 4.1.1 or higher.
4. GSL version 1.16 or higher.
5. Boost library version 1.53 or higher.
6. OpenGL Mathematics (GLM) version 0.9.6.1 or higher.
7. LaTeX for creating documentation.

The packages that are necessary to compile the core are checked during the configuration step. Python libraries are not checked.

1.3 Programming Policy

A strict programming policy is enforced to keep the code clean and rational. This policy is strictly enforced in particular due to usage of such an obscure, overloaded and heavy language as C++.

1.4 Development Policy

Chapter 2

Ice Element Location and Position

2.1 Introduction

Let an ice element be considered as a flat (2D) rigid body in a form of convex polygon moving on a spherical surface. The goals of this chapter are

- Find a suitable description of position and orientation of the ice element;
- Derive position integraion equations;
- Build an integration numerical algorithm to update position and orientation of the ice element;
- Describe coordinate transformations used in the model.

2.2 Equivalence to 3D Rigid Body Rotations

It is easy to see (but not so easy to find out) that the motion of an ice element on the surface of a sphere is equivalent to a rotation of a rigid body around the center of the sphere. Such a rigid body can be considered to be a pendulum with a massive weight to be a material polygon which center of mass is connected to the center of the sphere by a massless rod. The center of the sphere does not move and is a center of the rotation. This set up is shown in Figure. 2.1.

It also can be visualized as a rotation of the rigid sphere shell around its center with all the mass concentrated inside a single polygon patch – an ice element.

The representation of an ice element as a 3D constrained rotating rigid body provides a useful instrument to describe exact position and orientation of an ice element. The rigid body rotation around a point can be described by unit quaternions a.k.a versors [Graf, 2008, Waldvogel, 2008, Arribas et al., 2006, Nemes and Mikóczi, 2013]. The main advantage of using quaternions instead of, say, spherical coordinates with a rotation angle is that every point on a sphere is treated exactly the same way as any other point. The density of coordinate lines do not change from point to point and no critical points are present. The quaternion representation also allows to perfom many coordinate transformations without

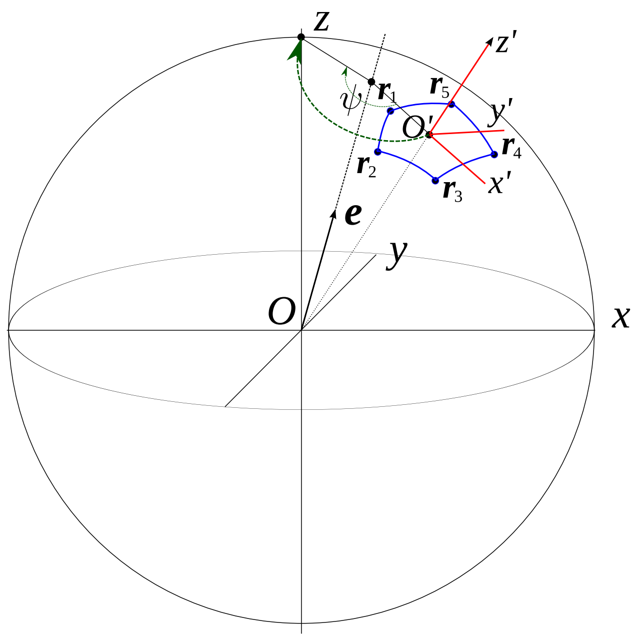

Figure 2.1: Coordinates

involving trigonometry, store orientation information very compactly, and the formulas of all necessary operations are simple and short. **A single quaternion is sufficient to uniqly describe the position and orientation of an ice element**.

2.3 Coordinate Systems and Frames

When describing different values such as positions or vectors we use 2 different types of reference points or “frames” depending on convenience:

- Global Frame, connected to the Earth and rotating with the Earth;
- Local Frames, a reference point connected to a particular ice element.

For describing elements in Global Frame, we use 3 different coordinate system:

- Geographical coordinates (latitude φ and longitude λ). They used only for import or export elements and never used internally. We provide trasformation from and to geographical coordinates.
- Spherical coordinates ($\phi = \lambda$, $\theta = \pi/2 - \varphi$).
- 3D extrinsic Cartesian coordinates (x', y', z'). Its center coincides with the center of the Earth O, z' is directed along Earth’s axis of rotation towards the North, x' direction is a direction to 0 meridian, and y' direction is chosen perpendicular to x' and z' to form a right coordinate system. These coordinates satisfy the constrain ${x'}^2 + {y'}^2 + {z'}^2 = R^2$.

- 3D normalized (dimensionless) extrinsic Cartesian coordinates (x, y, z). Same as above but all projected to a unit sphere such that $x^2 + y^2 + z^2 = 1$. *These are the main global coordinates internally used in the Core for all the computations.*

We use Global Frame only when considering interaction between two ice elements. In all other cases we transform global coordinates into a local frame first and then produce the computations.

The transformation between extrinsic coordinates and geographic/spherical coordinates are performed using the following formulas:

$$\begin{cases} \theta = \pi/2 - \varphi, \\ \phi = \lambda. \end{cases} \qquad \begin{cases} x = \sin\theta\cos\phi, \\ y = \sin\theta\sin\phi, \\ z = \cos\theta. \end{cases} \tag{2.1}$$

Local coordinates (X, Y, Z) are the coordinates that have its center in the center of mass of the ice element. The axis are directed such that they would coincide with global frame normalized extrinsic coordinates (x, y, z) after application of the rotation described by the quaternion $\hat{\boldsymbol{q}}$ linked to the ice element.

Global extrinsic coordinates for any point $\vec{p}$ are restored from local coordinates $\vec{P}$ by applying the rotation matrix recovered from quaternion:

$$\vec{p} = \boldsymbol{R}(\hat{\boldsymbol{q}}) \cdot \vec{P} \tag{2.2}$$

Rotation matrix can be recovered from quaternion as described in Section B.2.

2.4 Initial Frame Relation

To relate two different frames, we have to introduce initial quaternion $\hat{\boldsymbol{q}}_0$. Later we will provide a first order differential equation for the evolution of the quaternion. However, we have to find this initial quaternion to be able to resolve the evolution.

We found a simple way to set this initial position quaternion. Initially, we have initial coordinates of the polygon representing the ice elements in some of Global Frame coordinates. To find initial quaternion we will do the following

1. Find center of mass of the polygon in global frame.
2. Find a quaternion such that when applied, it moves the center of mass into the North Pole. There are an infinite number of such rotation quaternions, so we pick one.
3. We introduce initial local coordinates based on this initial quaternion such that they will coincide with extrinsic coordinates after the transformation. All vertices coordinates $\vec{P}_i = (X_i, Y_i, Z_i)$ are computed then. They will never change with time.

There is an uncertainty in quaternion choice that initially would transpose the center of ice element into North pole. Let us choose the quaternion that transpose the center of the ice element as follows.

Suppose we have coordinates of the center of mass in Global Frame in Extrinsic coordinates: $\vec{c} = (c_x, c_y, c_z)$, then we restore the transformation as a rotation along big circle to match $\vec{c}$ with North Pole as follows:

$$\hat{\boldsymbol{q}}_{\boldsymbol{0}} = \cos\theta/2 + \hat{e}\sin\theta/2, \qquad \hat{e} = \vec{c}\times\hat{e_z}/|\vec{c}\times\hat{e_z}|, \quad \theta = \arccos c_z/|\vec{c}|. \tag{2.3}$$

θ is spherical coordinates of $\vec{c}$. The rotation axis is perpendicular to $\vec{c}$ and direction to the North. Reminder: we use a convention that vectors and scalars are all a subset of quaternions and can be added to each other to make a new quaternion.

2.5 Position Update Equation

Suppose we know the vector of angular velocity of our "pendulum" $\vec{\omega}$. Please, notice that this is 3D angular velocity vector, not a rotation rate of the ice element. Later we will see how to calculate it from the velocity and rotation rate of the ice element itself. In this case we have the following expression for updating orientation quaternion

$$\dot{\hat{\boldsymbol{q}}} = \frac{1}{2}\vec{\omega}\circ\hat{\boldsymbol{q}}. \tag{2.4}$$

Where $\dot{q} = dq/dt$. Let $\vec{\Omega}$ be the same angular velocity vector expressed in Local frame coordinates. Using (B.10) we can derive the main position equation:

$$\dot{\hat{\boldsymbol{q}}} = \frac{1}{2}\hat{\boldsymbol{q}}\circ\vec{\Omega}, \tag{2.5}$$

While developing COUπ we created a very neat scheme to solve this type of equations. This scheme will be described here later based on COUπ documentation.

2.6 Placement Numerical Integration

Let us describe the numerical solution of the equation (2.5).

Let $\vec{\Omega}^{n+1/2} = \vec{\Omega}(t+\Delta t/2)$ (and this value we will actually find on the dynamic stage), $\hat{\boldsymbol{q}}^n = \hat{\boldsymbol{q}}(t)$, $\hat{\boldsymbol{q}}^{n+1} = \hat{\boldsymbol{q}}(t+\Delta t)$. The 2nd order scheme for the equation above will be

$$\hat{\boldsymbol{q}}^{n+1} = \hat{\boldsymbol{q}}^n + \frac{\Delta t}{4}(\hat{\boldsymbol{q}}^n + \hat{\boldsymbol{q}}^{n+1})\circ\vec{\Omega}^{n+1/2}. \tag{2.6}$$

Collecting all next step values on the left side and taking into account that quaternion product is not commutative, we have

$$\hat{\boldsymbol{q}}^{n+1}\circ\left(1 - \frac{\Delta t}{4}\vec{\Omega}\right) = \hat{\boldsymbol{q}}^n\circ\left(1 + \frac{\Delta t}{4}\vec{\Omega}\right). \tag{2.7}$$

We removed index $n+1/2$ from $\vec{\Omega}$ to make it easier to read. Brackets in (2.7) are conjugate quaternions. Using relation $\hat{\boldsymbol{q}}\circ\tilde{\hat{\boldsymbol{q}}} = |q|^2$ we have

$$\hat{\boldsymbol{q}}^{n+1}\circ\left(1 - \frac{\Delta t}{4}\vec{\Omega}\right)\circ\left(1 + \frac{\Delta t}{4}\vec{\Omega}\right) = \hat{\boldsymbol{q}}^n\circ\left(1 + \frac{\Delta t}{4}\vec{\Omega}\right)^2. \tag{2.8}$$

$$\hat{\boldsymbol{q}}^{n+1}\left(1+\frac{\Omega^2\,\Delta t^2}{16}\right) = \hat{\boldsymbol{q}}^{n}\circ\left(1+\frac{\Delta t}{4}\vec{\Omega}\right)^2 . \tag{2.9}$$

$$\hat{\boldsymbol{q}}^{n+1} = \hat{\boldsymbol{q}}^{n}\circ\left(1+\frac{\Delta t}{4}\vec{\Omega}\right)^2 \Big/ \left(1+\frac{\Omega^2\,\Delta t^2}{16}\right) \tag{2.10}$$

For any vector $\vec{a}$ using product Eq. (B.3) we have

$$(1+\vec{a})^2 = (1+\vec{a})\circ(1+\vec{a}) = 1 - a^2 + 2\vec{a}. \tag{2.11}$$

Then, we have

$$\hat{\boldsymbol{q}}^{n+1} = \hat{\boldsymbol{q}}^{n}\circ\left(1-\frac{\Omega^2\,\Delta t}{16}+\frac{\Delta t}{2}\vec{\Omega}\right) \Big/ \left(1+\frac{\Omega^2\,\Delta t^2}{16}\right) \tag{2.12}$$

Let us introduce the following quaternion

$$\hat{\boldsymbol{p}} = \left(1-\frac{\Omega^2\,\Delta t}{16}+\frac{\Delta t}{2}\vec{\Omega}\right) \Big/ \left(1+\frac{\Omega^2\,\Delta t^2}{16}\right), \tag{2.13}$$

then integration formula will have the form

$$\hat{\boldsymbol{q}}^{n+1} = \hat{\boldsymbol{q}}^{n}\circ\hat{\boldsymbol{p}}. \tag{2.14}$$

Computation wise it is a significant improvement over [Walton and Braun, 1993] quaternion integration formula or trigonometric approach like in [Harada, 2008].

It is easy to see that $|\hat{\boldsymbol{p}}| = 1$ and hence this formula does not change the absolute value of unit quaternion. However, round-off errors might accumulate in time shifting the absolute value of $\hat{\boldsymbol{q}}$ from 1. To ensure we are working within unit quaternion algebra, a renormalization should be done occasionally

$$\hat{\boldsymbol{q}} \leftarrow \hat{\boldsymbol{q}}/|q|. \tag{2.15}$$

Chapter 3

Dynamics

3.1 Introduction

The main problem in dynamics is to calculate 3D "pendulum" angular velocity $\vec{\Omega}$ in ice-element reference frame. This vector can be calculated exactly using 3D dynamics of a rigid body with constraints. However, it is not necessary. A much simpler way should be sufficient if we suppose that the ice-element size is much smaller than the radius of the Earth. This is equivalent to the assumption that the ice element is flat.

In spite of this assumptions we are not solving 2D cartesian problem. All we are saying is that we are considering the dynamics on a sphere with locally flat ice elements.

3.2 Dynamics in ice-element reference frame

As ice-element reference frame is a standard cartesian coordinates and we assume that the ice-element is a flat 2D ridig body, its velocity and rotation rate can be calculated simply by the following equations

$$\dot{V}_x = F_x/m, \qquad \dot{V}_y = F_y/m, \qquad \dot{\Omega}_z = N/I, \tag{3.1}$$

where (F_x, F_y) are external forces that include all driving forces, integral forces from interaction with other ice-elements, and Earth Coriolis force, m is the total mass of the ice element, I is a moment of inertia around Z axis, and N is a total torque calculated in the center of mass $\vec{C}$ that includes a torque from interaction with other ice elements and possible torque that can be calculated from the integral $(\nabla \times F)_z$ of the driving forces over the ice element (TODO: discuss with Jenny).

3.3 Restoring 3D angular velocity

It is pretty clear that if we know V_x, V_y, and Ω_z, we can restore $\vec{\Omega}$ as follows:

$$\vec{\Omega} = (-V_y/R, V_x/R, \Omega_z)^T. \tag{3.2}$$

Indeed, V_x is directed normal to Z and tends to rotate the "pendulum" around Y axis with V_x/R rate. The rotation direction is "right" and that is exactly Ω_y. Same considerations show that $|\Omega_x| = V_y/R$ and we need to choose "-" as a sign to take into account the right direction.

Taking this into account, we can rewrite the dynamics ecuations for global angular velocity vector in Local frame:

$$\dot{\vec{\Omega}} = \vec{H} = \left(-\frac{F_y}{mR}, \frac{F_x}{mR}, \frac{N}{I}\right)^T . \tag{3.3}$$

3.4 Dynamics Integration

A straightforward 2nd order in time scheme is applicable for Eq. (3.3):

$$\begin{cases} \Omega_x^{n+1/2} = \Omega_x^{n-1/2} - \Delta t\, F_y^n/mR, \\ \Omega_y^{n+1/2} = \Omega_y^{n-1/2} + \Delta t\, F_x^n/mR, \\ \Omega_z^{n+1/2} = \Omega_z^{n-1/2} + \Delta t\, N^n/I \end{cases} \tag{3.4}$$

Or in vector form:

$$\vec{\Omega}^{n+1/2} = \vec{\Omega}^{n-1/2} + \Delta t\, \vec{H}^n. \tag{3.5}$$

Thus, the numerical scheme for integration of motion of the ice element is completed in terms of rotational quaternions.

Chapter 4

Ice Element Polygon

4.1 Introduction

Each ice element is represented by simple convex polygon. Siku core accepts polygons with many precomputed values described in this chapter. Thus, all following computations are performed on Python side of the model.

Initially, the polygon is given as a set of its extrinsic Cartesian coordinates in Global Frame. If the coordinates are given in Geographic coordinates, they are recalculated to extrincsic Cartesian coordinates using (2.1). Thus, we just suppose here that all vertices are given in Global frame as a set of extrinsic Cartesian radius-vectors: $\{\vec{p}_j\}$, $j = 0, 1, \ldots N-1$, total N vertices.

4.2 Center of Mass, Area, Moment of Inertia

The center of mass $\vec{c}$ (CM) is a center of local Frame. Given coordinates of vertices, the center of mass is calculated in assumption that the polygon is flat. Otherwise, complicated intergration would be necessary to accurate computations.

First, we introduce geometrical pseudo-center $\vec{q}$ by a formula:

$$\vec{q} = \sum_j \vec{p}_j / N. \tag{4.1}$$

Although, it is not a CM, it is a convenient point that resides inside the polygon and allows to split it in triangles as shown in Fig. 4.1.

Now we can calculate the centers of mass of each triangle as

$$\vec{c}_i = (\vec{p}_i + \vec{p}_{(i+1)'} + \vec{q})/3, \qquad (i+1)' = (i+1) \bmod N. \tag{4.2}$$

And areas A_i of each triangle and total area A.

$$A_i = |(\vec{p}_i - \vec{q}) \times (\vec{p}_{(i+1)'} - \vec{q})|, \qquad A = \sum_{i=0}^{N-1} A_i. \tag{4.3}$$

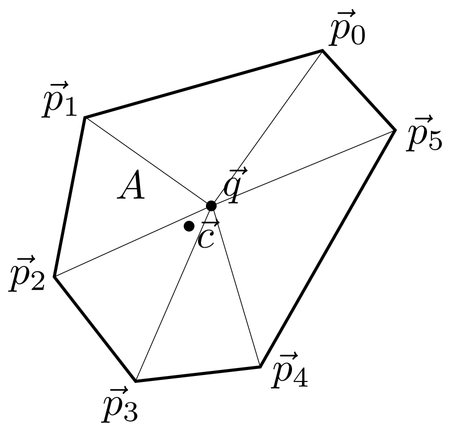

Figure 4.1: Polygon main properties computations

Now, the center of mass can be computed as a sum of area-weighted center of masses for each triangles:

$$\vec{c} = \sum_{i=0}^{N-1} A_i \vec{c}_i / A. \tag{4.4}$$

Total geometric moment of inertia (it is a moment of inertia divided by mass) around center of mass is computed by same weighted approach:

$$I/m = \sum_{i=0}^{N-1} A_i \left((\vec{p}_i - \vec{c})^2 + (\vec{p}_{(i+1)'} - \vec{c}) \cdot (\vec{p}_i - \vec{c}) + (\vec{p}_{(i+1)'} - \vec{c})^2 \right) / (6A). \tag{4.5}$$

4.3 Mass

Total mass need to be calculated up to at each timestep using the ice thickness distribution function $g(h)$. General formula is

$$m = \sum_{k=0}^{K-1} g(h_k) h_k \rho_k A, \tag{4.6}$$

where ρ_k is the density of ice of thickness h_k.

Chapter 5

Wind Interpolation

5.1 Introduction

The wind data is given in a mesh and, possibly, in some set of random points. We have to be able to restore the proper wind vector for any ice element on a sphere. We also would like to have a few analytical testing fields that we can use. This chapter describes all that related to the wind.

5.2 Analytical Testing Fields

First field to use is taken from [Fuselier and Wright, 2009], field 1, pp. 3233–3234.

5.3 NCEP/NCAR Reanalysis grids

Most part of testing and generic [?] wind interpolations is based on NCEP/NCAR Reanalysis data grids.

The data is represented as a values distributed upon the regular grid in latitude-longitude coordinates.

5.3.1 Basic interpolation method

As the grid is guaranteed to be regular and the wind field itself is pretty smoth - basic interpolation is made as if it is simple Cartesian coordinate system. The simpliest method to obtain the interpolated value in two dimensional space is bilinear interpolation. This method is based on the assumption that the value should smothly change along the distance between two known values. Saying 'smothly' means that the interpolated value should be calculated with the heuristic function, which defines the proportion law. The methed consists of next steps:

1. Constructing regular grid with all needed values in the nodes.

2. Find the grid cell, that matches to interpolation value position.

3. With heuristic proportion - calculating two additional values, tha are located on the same lattitude with interpolated value, but at the same time on cell‘ right and left boarders..
4. With that very heuristic propotion - calculating the value we were searching for.

This procedure means that proportion function shall be called three times in a raw. More then that - in most cases the result that is based on 'same latitude' additionel boarder values will differ from the result based on 'same longitude' additional boarder values.

5.3.2 Proportion functions

All circumstances described in previous section mean that proportion function should be as simple as possible, but at the same time it should preserve such field attributes as curl and divergence.

For satisfying such various conditions we provide several functions, that bestly suit for differents cases.

- Generic proportion and interpoation.

  The wind vector is represented as range-azimuth pair. Both values are considered to change linearily.

  $$|\vec{V_t}| = (1-t)|\vec{V_1}| + t|\vec{V_2}|. \tag{5.1}$$

  $$\phi_t = (1-t)\phi_1 + t\phi_2. \tag{5.2}$$

  Where $|\vec{V_t}|$ is interpolated velocity vector, ϕ is it‘s azimuth, t is scalar value ($t \in [0, 1]$)

- Simple proportion and interpolation.

  The wind vector is represented as east-north components pair. Both values are considered to change linearily.

  $$\vec{V_t} = (1-t)\vec{V_1} + t\vec{V_2}. \tag{5.3}$$

- SLERP proportion and interpolation.

  According to https://en.wikipedia.org/wiki/Slerp.

  $$\vec{V_t} = \frac{\sin[(1-t)\Omega]}{\sin\Omega}\vec{V_1} + \frac{\sin[t\Omega]}{\sin\Omega}\vec{V_2}. \tag{5.4}$$

  Where Ω is the angle between $\vec{V_1}$ and $\vec{V_2}$

5.4 Advanced interpolation (expected soon)

For more accurate and meaningfull interpolation several more complicated methods are going to be provided.

The main aim is to take into account such parameters as wind‘s first (and maybe second) derivative, that could be calculadet in the nodes, and additional random distributed wind data from various other sources.

Chapter 6

Force Balance on Ice Elements

During the Arctic Ice Dynamics Joint Experiment (AIDJEX) it was found that long term mean ice motion, in the summer when ice concentration is low, results from the balance of four forces on the ice [?, ?]. In free drift the significant forces on the ice are wind stress, ocean stress, the Coriolis force and acceleration down the sea surface tilt. Inertial terms were found to be an order of magnitude smaller than these on daily or longer time scales [?], however are important on shorter time scales and accumulative impact of the inertial motion of the ocean and ice maybe significant in the sea ice mass balance [?]. The residual term in the measured force balance is found to be comparable to the Coriolis force, and an order of magnitude smaller than the wind stress [?]. This residual is attributed to interactions between ice floes. In areas of ice convergence, resistance to compression becomes important, and ice can no longer be considered to be in free drift. For climatological studies small time scale forcing may be ignored. Contributions to the force balance from tidal motion, pressure gradients and inertial terms are considered to be small when averaged over time scales greater than a day [?]. For short term ice drift estimation, and in regions with strong tides the tidal and inertial terms may become important.

The force balance on the ice, per unit area, is given by: *** This needs to be replaced with force balance on an ice element. It needs to be integrated over the element area for surface forces and volume for body forces. We need to replace $\nabla \cdot \boldsymbol{\sigma}$ with contact forces.***.

$$\frac{\partial m\vec{U}}{\partial t} + \nabla \cdot \left(m\vec{U}\vec{U} \right) = \boldsymbol{\tau}_a + \boldsymbol{\tau}_w - mf\hat{\vec{k}} \times \vec{U} - mg\nabla H + \nabla \cdot \boldsymbol{\sigma}, \tag{6.1}$$

where m is the ice mass, $\vec{U}$ is the ice velocity, $\boldsymbol{\tau}_a$ and $\boldsymbol{\tau}_w$ are the wind stress and ocean stress on the ice, f is the Coriolis parameter, $\hat{\vec{k}}$ is a unit vector normal to the surface, g is the acceleration due to gravity and H is the sea surface dynamic height. The remaining term $\nabla \cdot \boldsymbol{\sigma}$ is a force due to sub-scale interactions between ice floes. A constitutive law defining the ice stress $\boldsymbol{\sigma}$ will be outlined in section ??. By scale analysis [?] it can be shown that the advection of momentum, $\nabla \cdot (m\vec{U}\vec{U})$ is two orders of magnitude smaller than the coriolis and gravitational terms and may be neglected. The inertial term $\frac{\partial m\vec{U}}{\partial t}$ is also small on time scales greater than a day. The momentum may be found from the balance between external forces and $\nabla \cdot \boldsymbol{\sigma}$, which is the form [?] take for their model. The two largest terms, $\boldsymbol{\tau}_a$ and $\boldsymbol{\tau}_w$, are of the order $1\,\mathrm{kgm^{-1}s^{-2}}$, and tend to oppose each other. The Coriolis, gravitational and interaction terms are of the same order of magnitude, and typically an order of magnitude smaller than $\boldsymbol{\tau}_a$ and $\boldsymbol{\tau}_w$.

*** Need to re-write the above to reflect the shorter timescale we will resolve with Siku. And to write in terms of contact mechanics rather than continuum form. We also need to discuss inertial motion. The ice/ocean does manifest inertial oscillations in response to storms, it is the motion of the massive ocean that is causing the ice to oscillate. Which is not possible to model with out a coupled ice/ocean model. ***

6.1 Surface Forcing

The force balance on an ice element includes stresses to the upper (atmosphere) and lower (ocean) surfaces that must be prescribed.

The stress applied over a unit area of ice by the wind and ocean current is modelled with a quadratic boundary layer model [?, ?, ?]. Ice velocity is an order of magnitude smaller than the wind velocity and insignificant when calculating the wind stress over the ice,

$$\boldsymbol{\tau}_a = \rho_a C_a |\vec{U}_a| \left(\vec{U}_a \cos\theta_a + \hat{\vec{k}} \times \vec{U}_a \sin\theta_a \right), \tag{6.2}$$

$$\boldsymbol{\tau}_w = \rho_w C_w |\vec{U}_w - \vec{U}| \left(\left(\vec{U}_w - \vec{U} \right) \cos\theta_w + \left(\vec{U}_w - \vec{U} \right) \hat{\vec{k}} \times \sin\theta_w \right). \tag{6.3}$$

The drag laws require the air velocity, $\vec{U}_a$, at a known height in the atmosphere and the ocean current, $\vec{U}_w$, usually below the surface Ekman layer, to be known. It is possible to take these as geostrophic velocities, though they could equally well be provided at heights within the boundary layer by atmospheric and oceanic models. The other variables are the density of air, ρ_a; density of water, ρ_w; the drag coefficients for the air and water interfaces, C_a and C_w; and the turning angles across the boundary layers, θ_a and θ_w. The magnitude of the drag parameters and turning angle is dependent upon the height (depth) that winds (currents) are provided for.

*** Let's start with the simplist implimentation for drag coefficients and turning angles, assuming they are constant. C_a and C_w could be modeled as a function of ice morphology following [?], and they can be adjusted to better fit observed ice drift following [?]. I also need to consider a version of the atmospheric stress parameterisation that allows for an unstable atmosphere boundary layer. This will take me a couple of days to sort out in my head. So beware, we may be replacing the wind stress equation with something else.

For testing it is appropriate to start with the default values for the parameterisations. These are $C_w = 0.0045$, $\theta_w = 0$, $C_a = 0.0016$ and $\theta_a = 0$. We are assuming the velocities are sufficiently close to the surface that Eckmann turning is negligable.

6.2 Data Input

Wind and ocean current data can be sourced from various agencies. Either forecast model output can be used or reanalysis fields. Provided data is provided on a known grid SLERP (see section ??) will interpolate the fields to ice element locations.

6.2.1 Winds

Siku is set up to run with 10 m winds by default. It is possible to use geostropic winds or winds from different atmospheric levels, however to do this the drag parameterisation and turning angles will have to be adjusted in Siku's calculation of surface stress. *** ANTON: I assume we will have a lookup table variables are read in from where these can be set ***.

Example wind fields are provided with the Siku distribution for testing purposes.

6.2.2 Ocean Currents

*** Will provide information as soon as we have narrowed down which fields we shall use from Hycom or RASM. We should also include a note about adding tides to the currents, which could be provided by [**?**] for example. ***

For testing purposes one can set the ocean currents to zero. In this can the ice is moving over a stationary ocean that provides drag.

Chapter 7

Diagnostics

Diagnostics is a facility to calculate and output some values from the model on a set of coordinates.

For each diagnostics, the user registers a function in python script where the data from diagnostics will be transferred to. Together with this function, the user provides and registers set of points, a.k.a. “mesh” where the data will be calculated. The user also schedules the runs such that diagnostics will be performed only when necessary, not at each time step.

Chapter 8

Random points generation

8.1 Problem Statement

We need to generate a set of points on sphere that will serve the role of nuclei for Voronoi tesselation procedure. The set needs to satisfy the following conditions:

1. The point set need to be randomly generated with some distribution (resolution) function over sphere. This means that some areas will have finer resolution than others.

2. The point set need to have blue noise characteristics. This means that the minimum distance between two points cannot exceed some limit value $d_{\min}$. This limit value is also a function of location.

3. The boundaries need to be also correctly represented. This means that the set of points need to contain explicitly the points from the boundaries. The resolution near the boundaries should be set by resolution function.

The problem is not trivial. The standard approaches for generation of blue noise are not always applicable here. On the other hand, we do not need too sophisticated approach as the statistical quality of the final set is not that critical.

The points are generated with `mesh/gen_points.py` script that is maintained to satisfy the procedure described in this chapter.

8.2 Random points on sphere

8.2.1 Uniform distribution on sphere

To generate a uniform point distribution a trivial random number generation in the cube (x, y, z) does not work. To pick a random point on sphere we use the method by [Marsaglia, 1972]. The random point (x, y, z) on a sphere is generated using the following rule:

1. Generate X and Y, uniformly distributed on a segment (-1,1).

2. Reject all the points that $X^2 + Y^2 \geq 1$.

3. Generate a point on a sphere from the remaining points as follows:

$$\begin{aligned} x &= 2X\sqrt{1-X^2-Y^2}, \\ y &= 2Y\sqrt{1-X^2-Y^2}, \\ z &= 1-2(X^2+Y^2) \end{aligned} \tag{8.1}$$

8.2.2 Blue noise

The blue noise point generation is when the points are almost randomly generated but with additional condition that there is no two points that are closer than a particular distance d_{min}. The simplest way to generate blue noise is to remove a subset of the points from the original set such that a new set will satisfy the blue noise condtion. We proposed the following algorithm for this.

1. Sort all the points by x coordinates (we will compare the projection of the points on x-axis).
2. Form a new empty list that will contain the blue noise set. Add the first point from the original list to it.
3. Traverse the original list starting from its second point. For each original point check the points from the new list backwards until the projection distance is smaller than d_{min}. If no points are actually closer than d_{min}, add original point to the new list. Otherwise, skip it.

The new list will obviously satisfy the blue noise condition.

8.2.3 Large voids

There is no garantee that some points will be too isolated from the set with much voids around. This problem can be solved by adding points in the voids. However, we will not address it for now.

8.2.4 Changing the distribution

However, we need to generate the point set with varying distribution function.

The first and simplest approach would be to generate the point set with the finest resolution possible. Then we can apply the blue noise filter technique described in Sec. 8.2.2. However, this can be extremely slow even for not so fine resolution.

Instead, we can set the areas with different resolutions and the simplest way is to generate uniform resolution in that areas and then apply filter to the whole globe.

8.2.5 Uniform distribution is selected region

The method from Sec. 8.2.1 cannot be directly apply. However, we can use different approach. Technically, the uniform distribution over the globe can be generated by

$$\theta = 2\pi u, \quad \phi = \arccos(2v - 1), \tag{8.2}$$

where u and v are random values on $(0, 1)$.

Appendix A

Formulas Summary

Dynamics

$$\dot{\vec{\Omega}} = \vec{H} = \left(-\frac{F_y}{mR}, \frac{F_x}{mR}, \frac{N}{I}\right)^T. \tag{A.1}$$

$$\vec{\Omega}^{n+1/2} = \vec{\Omega}^{n-1/2} + \Delta t\, \vec{H}^n. \tag{A.2}$$

Placement

$$\dot{\hat{\boldsymbol{q}}} = \frac{1}{2}\hat{\boldsymbol{q}} \circ \vec{\Omega}, \tag{A.3}$$

$$\hat{\boldsymbol{p}} = \left(1 - \frac{\Omega^2\,\Delta t}{16} + \frac{\Delta t}{2}\,\vec{\Omega}\right) / \left(1 + \frac{\Omega^2\,\Delta t^2}{16}\right), \tag{A.4}$$

$$\hat{\boldsymbol{q}}^{n+1} = \hat{\boldsymbol{q}}^n \circ \hat{\boldsymbol{p}}. \tag{A.5}$$

Appendix B

Quaternions and Rotations

B.1 Briefly about Quaternions

[Graf, 2008] gives a good introduction to quaternions and their application to represent rigid body rotations. Here we will represent a few facts that will be useful in future discussions. Most of the proofs will be omitted.

B.1.1 Definitions

A quaternion is a number system that extends the complex numbers. Quaternions are 4-dimensional linear vector space over real numbers with scalar product and quaternion product defined on it. Algebraic definition similar to complex numbers is the following. A quaternion $\boldsymbol{q}$ is defined as

$$\boldsymbol{q} = q_0 + q_1 i + q_2 j + q_3 k, \qquad q_l \in \mathbb{R}, \quad l = 0, 1, 2, 3, \tag{B.1}$$

where $i^2 = -1$ (same as with complex numbers), $j^2 = k^2 = ijk = -1$. The multiplication of quaternion is non-commutative. That is the main distinction of them with complex numbers.

q_0 is said to be a real value and q_1, q_2, and q_3 to be imaginary values. The quaternion product $\circ$ of two quaternions can be algebraically derived from these rules.

B.1.2 Vector representation

Quaternions can be represented in different form as a pair of a number and a vector as follows

$$\boldsymbol{q} = (q_0, \vec{q}), \qquad \operatorname{Re} \boldsymbol{q} = q_0, \qquad \operatorname{Im} \boldsymbol{q} = \vec{q}. \tag{B.2}$$

The quaternion product of two quaternions $\boldsymbol{q}$ and $\boldsymbol{p}$ can be defined in this notation as follows

$$\boldsymbol{q} \circ \boldsymbol{p} = (q_0 p_0 - \vec{q} \cdot \vec{p}, q_0 \vec{p} + p_0 \vec{q} + \vec{q} \times \vec{p}). \tag{B.3}$$

B.1.3 Vectors as quaternions

Vectors can be considered to be quaternions with 0 real part. For two vectors $\vec{a}$ and $\vec{b}$ we have

$$\vec{a} \circ \vec{b} = (-\vec{a} \cdot \vec{b}, \vec{a} \times \vec{b}). \tag{B.4}$$

Thus, quaternion product contains both dot- and cross-products of vectors.

B.1.4 Norm

Norm of the quaternion is defined as follows

$$|\boldsymbol{q}| = \left(q_0^2 + \vec{q} \cdot \vec{q}\right)^{1/2} = \left(q_0^2 + q_1^2 + q_2^2 + q_3^2\right)^{1/2}. \tag{B.5}$$

It can be shown that

$$|\boldsymbol{q} \circ \boldsymbol{p}| = |\boldsymbol{q}||\boldsymbol{p}|. \tag{B.6}$$

B.1.5 Conjugate

The conjugate $\bar{\boldsymbol{q}}$ of quaternion $\boldsymbol{q}$ is defined as

$$\bar{\boldsymbol{q}} = (q_0, -\vec{q}). \tag{B.7}$$

It can be shown that

$$\bar{\boldsymbol{q}} \circ \boldsymbol{q} = \boldsymbol{q} \circ \bar{\boldsymbol{q}} = |q|^2. \tag{B.8}$$

B.1.6 Unit quaternions

Thus, unit quaternions such that $|q| = 1$ form a subalgebra of quaternions over quaternion product operation. It can be shown that the quaternion is unit if and only if it can be represented in the following form

$$\boldsymbol{q} = \left(\cos\frac{\psi}{2}, \sin\frac{\psi}{2}\,\vec{e}\right), \qquad |\vec{e}| = 1. \tag{B.9}$$

ψ is known as “rotation angle” and unit vector $\vec{e}$ is known as “rotation axis vector”.

B.1.7 Coordinate transformations using quaternions

Let $\vec{a}$ be a vector represented in the Global frame $\{x, y, z\}$. Let $\vec{A}$ be the same vector represented in local coordinate system $\{X, Y, Z\}$. In this case it can be shown that coordinate transformations between two systems can be expressed as follows:

$$\vec{A} = \bar{\boldsymbol{q}} \circ \vec{a} \circ \boldsymbol{q}, \qquad \vec{a} = \boldsymbol{q} \circ \vec{A} \circ \bar{\boldsymbol{q}} \tag{B.10}$$

B.2 Rotation Matrix and Positions

An important problem (mostly for visualization) is to determine spherical coordinates (ϕ, λ) of an element polygon vertex by quaternion $\boldsymbol{q}$ of the polygon and vertex local coordinates (X_i, Y_i). When local coordinates of the vertex $\vec{P} = (X, Y, Z)^T$ are known, the global coordinates can be found as follows:

$$\vec{p} = \boldsymbol{R} \cdot \vec{P}, \tag{B.11}$$

where $\boldsymbol{R}$ is 3D rotational matrix restored from quaternion. The formulas can be found in many places, for example in a general introduction [Wikipedia, 2012]. For rotation matrix R and quaternion $\hat{q}$ with elements (s, v_x, v_y, v_z) we have

$$R(\hat{q}) = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{pmatrix}, \tag{B.12}$$

or in a more computationally convenient way

$$R(\hat{q}) = \begin{pmatrix} 1 - 2q_2^2 - 2q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & 1 - 2q_1^2 - 2q_3^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & 1 - 2q_1^2 - 2q_2^2 \end{pmatrix} \tag{B.13}$$

The most efficient way to perfom these computations was found in [Eberly, 2008]. We just use standard libraries available for these transformation.

Bibliography

[Arribas et al., 2006] Arribas, M., Elipe, A., and Palacios, M. (2006). Quaternions and the rotation of a rigid body. *Celestial Mechanics and Dynamical Astronomy*, 96(3-4):239–251.

[Eberly, 2008] Eberly, D. (2008). Rotation representation and performance issues. Technical report, Geometric Tools, LLC, www.geometrictools.com. From documentation to Wild Magic library.

[Fuselier and Wright, 2009] Fuselier, E. J. and Wright, G. B. (2009). Stability and error estimates for vector field interpolation and decomposition on the sphere with rbfs. *SIAM Journal on Numerical Analysis*, 47(5):3213–3239.

[Graf, 2008] Graf, B. (2008). Quaternions and dynamics. *arXiv preprint arXiv:0811.2889*.

[Harada, 2008] Harada, T. (2008). Real-time rigid body simulation on GPUs. In Nguyen, H., editor, *GPU Gems 3*, pages 611–631. Addison-Wesley, NVIDIA.

[Marsaglia, 1972] Marsaglia, G. (1972). Choosing a point from the surface of a sphere. *Ann. Math. Statist.*, 43(2):645–646.

[Nemes and Mikóczi, 2013] Nemes, F. and Mikóczi, B. (2013). Perturbed kepler problem in general relativity with quaternions. *International Journal of Modern Physics D*, 22(08).

[Waldvogel, 2008] Waldvogel, J. (2008). Quaternions for regularizing celestial mechanics: the right way. *Celestial Mechanics and Dynamical Astronomy*, 102(1-3):149–162.

[Walton and Braun, 1993] Walton, O. R. and Braun, R. L. (1993). Simulation of rotary-drum and repose tests for frictional spheres and rigid spere clusters. In *5-th Joint DOE/NSF workshop on flow of particulates and fluids*, Ithaca, NY, USA.

[Wikipedia, 2012] Wikipedia (2012). Quaternions and spacial rotationa. Wikipedia.org. at http://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation.