NUMERISCHE MODELLIERUNG VON MIKROSTRUKTUREN IN EIS

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

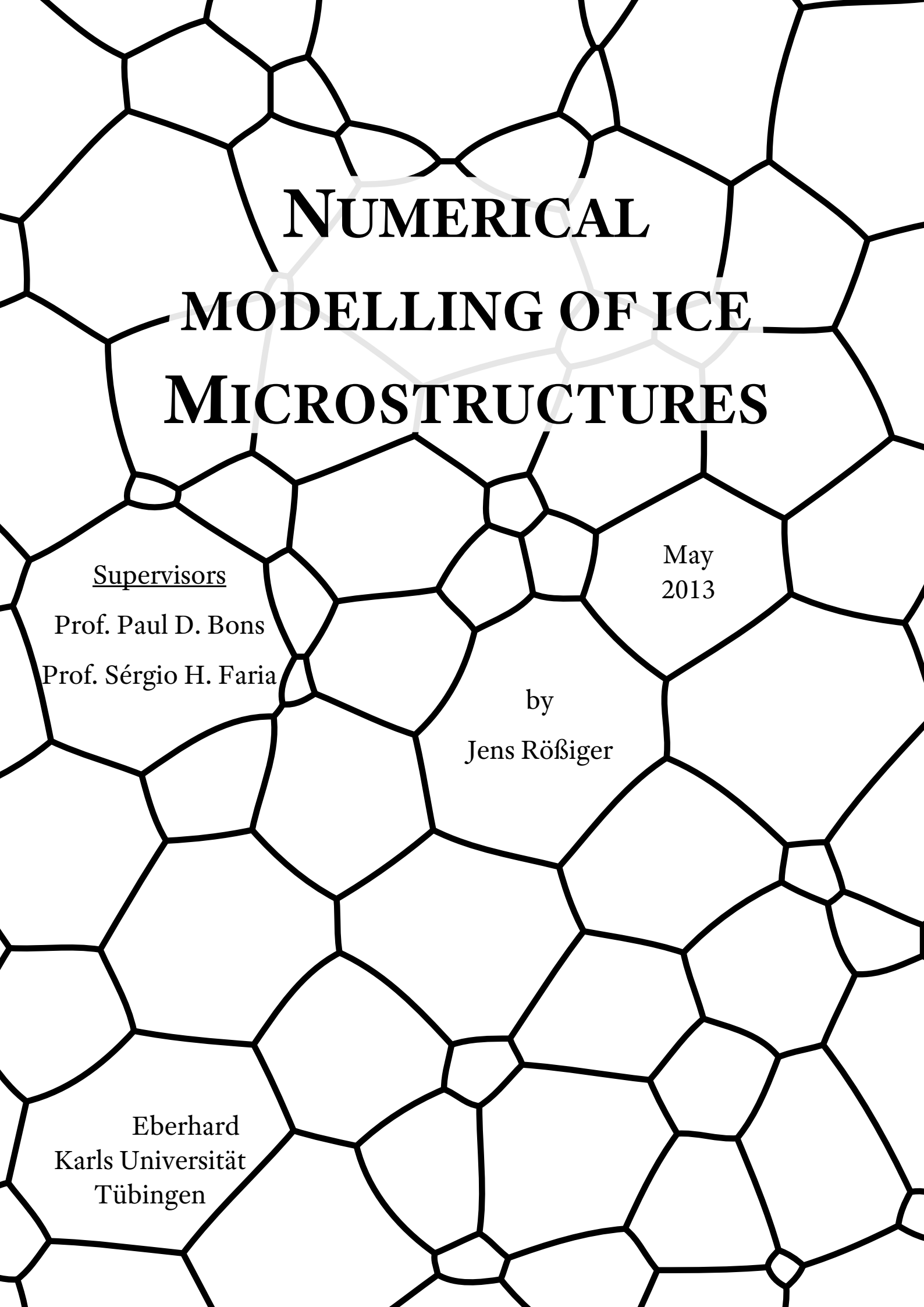
Jens Rößiger

aus Stuttgart

Tübingen

2013

Tag der mündlichen Qualifikation:	18.07.2013
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Paul D. Bons
2. Berichterstatter:	Prof. Dr. Sérgio H. Faria



NUMERICAL MODELLING OF ICE MICROSTRUCTURES

May
2013

Supervisors

Prof. Paul D. Bons

Prof. Sérgio H. Faria

by

Jens Rößiger

Eberhard
Karls Universität
Tübingen

Ich erkläre hiermit, dass ich die zur Promotion eingereichte Arbeit selbständig verfasst, nur die angegebenen Quellen und Hilfsmittel benutzt und wörtlich oder inhaltlich übernommene Stellen als solche gekennzeichnet habe. Ich erkläre, dass die Richtlinien zur Sicherung guter wissenschaftlicher Praxis der Universität Tübingen (Beschluss des Senats vom 25.5.2000) beachtet wurden. Ich versichere an Eides statt, dass diese Angaben wahr sind und dass ich nichts verschwiegen habe. Mir ist bekannt, dass die falsche Abgabe einer Versicherung an Eides statt mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft wird.

Tübingen, Mai 2013

JR

_____

Jens Rößiger

CONTENTS

ZUSAMMENFASSUNG

Für Klimarekonstruktionen, -hochrechnungen und andere wissenschaftliche Fragestellungen ist es von enormer Wichtigkeit in der Lage zu sein präzise Modelle für das Verhalten von Eis unter natürlichen Bedingungen aufstellen zu können. Diese beeinflussen zum Beispiel Aussagen über Meeresspiegelschwankungen und Interpretationen von Klimasignalen oder -veränderungen und deren Auswirkung auf die Eisschilde. Besseres Verständnis und erweiterte Formeln sind nötig um die Modellierung von Gletschern und Eisschilden zu verbessern. Das Ziel dieser Arbeit war es, numerische Modelle zu entwickeln die in der Lage sind Mikrostrukturen zu simulieren wie sie in Experimenten und der Natur beobachtet wurden. Nachdem viele neue Funktionen und Algorithmen für die Modellierplattform „Elle" entwickelt wurden, konnten die numerischen Experimente damit durchgeführt werden.

In der ersten Experimentserie wurden die Auswirkungen von statischem Kornwachstum mit gleichzeitig ablaufender Korngrößenverkleinerung durch dynamische Rekristallisation untersucht. Ein linearer Anstieg der Kornfläche wird von der etablierten Theorie für statisches Kornwachstum prognostiziert. Der Parameter *k* legt die Wachstumsgeschwindigkeit fest. Für *k* wird oft ein konstanter Wert eingesetzt welcher material- und temperaturabhängig ist. Die Simulationen zeigen jedoch, dass *k* von der Mikrostruktur abhängt und variieren kann während dieselbe sich entwickelt. Experimente die mit Ungleichgewichtsstrukturen starten liefern möglicherweise Wachstumsparameter (*k* und n, dem Wachstumsexponent), die größer oder kleiner sind als die, durch die Theorie, für Gleichgewichtsstrukturen prognostizierten. Mit der Mikrostruktur entwickelt sich auch *k* bis ein Gleichgewicht erreicht ist. Dadurch ergeben sich Auswirkungen auf die Bestimmung von *n* und damit auf den bestimmten, vorherrschenden Wachstumsprozess.

In der zweiten Experimentserie wurde eine zweite Phase dem System hinzugefügt. Die in Eis am häufigsten vorkommenden Sekundärphasen sind Luftblasen und Staubpartikel. In den oberen Schichten der Eisschilde sind Luftblasen sehr häufig. Die Simulationen zeigen, dass sie abhängig von ihrer Verteilung, Größe und Häufigkeit eine deutliche Auswirkung auf das Wachstumsverhalten der Eiskristalle haben. Dies wiederrum beeinflusst den Wachstumsparameter *k* und möglicherweise auch *n*. Durch die Ergebnisse konnten drei unterschiedliche Wachstumsregimes festgelegt werden. Im ersten Regime sind die meisten Korngrenzen noch blasenfrei und können ungehindert, ähnlich dem reinen Eis, wachsen. Im darauf folgenden, zweiten Regime werden mehr und mehr Korngrenzen von Blasen beeinflusst und das Wachstum verlangsamt sich. Im dritten und letzten Regime ist der Gleichgewichtszustand erreicht und die meisten Korngrenzen stehen im Kontakt mit Luftblasen.

In der letzten Experimentserie wurde visko-plastische Deformation dem System hinzugefügt, um das Fließen des Eises zu simulieren. Um die plastische Kristalldeformation des polykristallinen Aggregats zu simulieren kam die „Full-Field Theorie (FFT) zur Anwendung. Durch Dislokationen, die durch die Deformation hervorgerufen wurden, gelangte zusätzliche Verformungsenergie ins System. Diese zusätzliche, treibende Kraft bewirkt ein verstärktes Kornwachstum sowie eine

Regeneration des Kristallgitters. Durch sequenzielle Experimente mit unterschiedlicher Stärke dieser Effekte wurde der Einfluss derselben auf die Mikrostruktur bestimmt. Simulationen ohne Regenerationseffekte sind nur für sehr kleine Verformungen mit Experimenten vergleichbar. Um die Simulationen stärker zu deformieren ist es notwendig weitere Prozesse dem System hinzuzufügen welche die angestaute Energie wieder abbauen. Korngrenzmigration ist ein sehr effektiver. Durch überwachsen der hochenergetischen Bereiche werden Dislokationen aus dem System entfernt und ein neues, undeformiertes Kristallgitter geschaffen. Dislokationen bewegen sich auch von selbst und formen dabei entweder neue Korngrenzen durch Akkumulation vieler Dislokationen oder gegensätzliche Dislokationen löschen sich wieder aus. Dadurch regeneriert sich das Kristallgitter zum Teil. Die Experimente zeigen, dass realistische Simulationen auf dem Maßstab von Kristallkörnern nur durchgeführt werden können, wenn wirklich alle bedeutenden Prozesse mitwirken.

Die Ergebnisse dieser Arbeit sind in Form von drei Publikationen beigefügt. Zwei derselben sind bereits akzeptiert und publiziert, eine weitere wurde nach kleineren Korrekturen wieder eingereicht. Außerdem findet sich am Ende der Arbeit ein ausführlicher Anhang der die Teile der Simulationssoftware beschreibt welche während dem Projekt neu entwickelt wurden.

ABSTRACT

Accurate modelling of ice mechanical behaviour under natural conditions is important for climate reconstruction and prediction, as well as for other scientific questions. It influences estimates of sea level changes and interpretation of past climate variations or signals recorded in ice cores. Better insight into the behaviour and constitutive equations of ice is imperative to improve modelling of glaciers and ice sheets. The aim of this thesis was to develop numerical models to simulate the microstructural behaviour of ice, as observed in nature and experiments. Numerical simulations were carried out with the numerical modelling platform "Elle", for which many new routines and algorithms were developed and implemented in this project.

In a first series of models static grain growth and simultaneous grain-size reduction by rotational recrystallization was investigated. Well-established theory for static grain growth predicts a linear increase of the grain area with time for ice. The growth rate is then determined by the growth parameter *k*, which is commonly assumed to be a temperature and material-dependent constant. However, the simulations show that *k* also depends on the microstructure and can thus vary as the microstructure evolves. Experiments that start with non-equilibrium microstructures potentially yield growth parameters (*k* and the growth exponent, *n*) larger or smaller than theory predicts for equilibrium foam textures. As the microstructure evolves *k* also changes until a steady state is reached. This has an impact on the estimation of the growth exponent *n* in experiments and therefore the implied rate controlling process.

In the second series of simulations, a second phase was added to the system. The most common second phase in ice are air bubbles and small dust particles. In the upper part of ice sheets bubbles are abundant and simulations show that depending on their size, amount and distribution they have a major impact on the growth behaviour of the ice crystals. This in turn affects the growth parameter *k* and potentially *n*. Results revealed three distinct growth regimes. In the first regime most grain boundaries are bubble-free and can grow unhindered and similar to those in pure ice. That is followed by the transitional regime where more and more boundaries start to get in contact with the air bubbles and growth slows down. Finally a steady state is reached where most boundaries are affected by bubbles.

As a final project crystal-plastic deformation was included in the system to simulate flow of ice. Additional strain energy introduced by deformation-induced dislocations adds another driving force for recrystallization, resulting in recovery and enhanced grain-boundary migration. The Full-Field Theory (FFT) was used to simulate crystal-plastic deformation of polycrystalline ice. Controls on the microstructure were investigated by comparing results of simulations with different relative rates of recovery and grain-boundary migration. Recovery-free simulations are only comparable to experiments for very small amounts of strain. Applying more deformation to the system makes it necessary to add processes which dissipate internal energy from the system. Grain-boundary migration is one effective process since the boundaries sweep dislocations from the system and give rise to a recrystallized, undeformed lattice. Recovery is another important process since dislocations can also move, accumulate in sub-grain boundaries

and annihilate themselves by combining with their counterparts. The simulations show that realistic grain-scale simulations can only be achieved when all grain-scale processes are included.

In this thesis, the results of the research are included in the form of two accepted publications, one publication that is at the date of submission awaiting final approval after minor revisions and, finally, an extensive appendix that describes the new simulation software that was written and implemented during this project.

INTRODUCTION

The climate around the world is changing and currently many people are trying to understand the reasons, the origin and the consequences of these changes. From geological records we know that it already happened many times in the past [1,2]. Since the deposition of one meter of sediments usually takes from a few centuries up to thousands of years [3,4], these records usually do not provide data with adequate resolution. We have learned however that there were times in the past with no ice coverage at all and times with very large ice sheets [1,2,5]. Some have postulated that there were even times when the whole Earth was covered by ice, a so-called "snowball Earth" [6-8]. In cold regions, snow may also be considered sediment. Since the precipitation of snow is usually much higher than the sedimentation rate of rock sediments [9,10], climate records in snow and ice provide a higher resolution. Therefore, climate records from the polar ice sheets have offered the most detailed records of the last hundred thousands of years [11,12]. So far there is no known ice on Earth older than about one million years [13,14]. With the records derived from ice cores we are trying to better understand the last few climatic changes. Also the influence of humans, especially over the last few centuries [15], is much better recorded in ice sheets on Antarctica or Greenland than in the sedimentological record. The effect of climate change is clearly seen in the rapid retreat of the smaller glaciers, such as those in the Alps [16-18]. Photographs from a few decades ago in direct comparison with recent pictures reveal a massive retreat of most glaciers. However, to get good data and information from these ice records we have to understand how ice moves, flows, deforms, and reacts to changing conditions.

Ice can essentially be considered as both a mineral and a rock with their own particular properties. A summary of these is given in Petrenko & Whitworth [19], the material properties are also discussed in Schulson and Duval [20]. There are a few major differences when compared to most rocks on the surface of the Earth. The first one, and probably the most obvious, is that it consists of only one major phase, which is frozen water. However for many investigations this simplification cannot be used anymore, as ice contains small quantities of impurities (e.g. dust) and trapped air forms a major second phase in the upper hundreds of meters of ice sheets [21,22]. The second major difference is that on Earth ice is always very close to its melting point [23]. Assuming that the average geothermal gradient is about 30°C per kilometre and that the

average rock type in the earth's crust melts at 800°C [24], one would need to consider depths of at least 20-25km to find rocks that are comparable to ice on Earth's surface. As these rocks, ice is ductile and flows, albeit very slowly for human perception [25,26]. Since it does flow, the interpretation of records that are stored in an ice sheet becomes difficult. Many different parameters influence how it flows in different positions and depths of the ice sheet. Because of that, a linear or similar simple age versus depth relationship in ice cores does not suffice for meaningful climate record interpretations [11,27].

When dating ice cores, there are some well-known events that can easily be identified. Most of them are related to major volcanic eruptions. Large amounts of ash and tephra reach high levels in the atmosphere during such events and therefore get distributed around the globe. Even if the eruption was thousands of kilometres away there will be ash deposition in thin layers everywhere on Earth [28,29] and can often be identified in the polar ice records. Since they can be recognized and distinguished quite well because of different shapes of ash particles or tephra from different volcanic eruptions, these horizons are very good age markers [30-32]. However, this is not enough for detailed climate reconstructions from ice cores. First of all the eruptions do not happen regularly [32], and second, ages of these eruptions are only known with some (~10%) error [32,33]. Time of the eruption is only known accurately in historical times. For age reconstruction of the ice cores it would be perfect to develop an accurate model that describes the flow of ice on small scales. This would provide a tool to calculate age-depth relationships at the location of the drill core. However, computing power is not yet sufficient, nor is it possible (yet) to include all relevant processes and parameters in a single model. An overview of numerical approaches and methods to simulate deformation of ice at various scales is given in Montagnat, et al. [34].

THE ELLE SIMULATION FRAMEWORK

There are different approaches to set up a simulation. In continuous models, attributes and properties of the material are parameterised in equations [35-39]. To save computation time these parameters can also be averaged across certain areas or volumes. A different approach is a discrete model. The modelling framework Elle uses such an approach. During the whole simulation a discrete image of the microstructure itself is used [40-43]. Elle is modular and each process is separate. They can be combined as required for the experiments with a script file

that calls the different processes one after the other. Due to technical reasons they do not act on the microstructure simultaneously, but sequentially. This is permissible for very small time steps Elle offers different modelling methods. Many processes in Elle make use of the front-tracking method, where the change of boundaries between different regions (polygons) is tracked. Other processes utilise the finite-element, finite-difference or other methods.

To describe the system under consideration, Elle offers two different data layers. One consists of points that can be placed randomly or on a regular grid. They are not connected to each other and are called "u-nodes". The second layer consists of points, "b-nodes", that are connected to each other and define polygons, termed "flynns". Each b-node can have two or three neighbouring nodes. The flynns represent grains or areas while the b-nodes and their connections define the boundaries between these grains. Both layers can be used in conjunction with each other or separately.

For the first project the growth code based on boundary curvature already existed [44] but the combination with a code to reduce the grain size and simulate polygonisation had to be written. The existing code to split grains in two was found to be inadequate and was rewritten for the purposes of this project. For the second project, a routine for two-phase grain boundary migration was already available. However, it was created for the simulation of melt pockets in a crystalline material, which usually have low dihedral angles [45]. The code was not suitable for systems with high dihedral angles, such as air bubbles in ice, as all air from various bubbles would tend to "diffuse" into one big bubble. The routine had to be rewritten completely to be applicable to ice with air, with the major challenge being to preserve areas of individual bubbles. This is now done by tracking the area of each phase region (single flynns or connected clusters). Two interaction rules can be chosen. In the first, all second-phase regions are assumed connected, which is effectively equal to the routine of Becker et al. [45].In the second, second-phase regions maintain their own area, but can be merged with others when migration leads to their impingement.

So far no simulation has included effects of deformation and only incorporated the effects of polygonisation as an abstract mechanism. Dynamic recrystallization can affect the microstructure significantly by changing grain size, shape and crystallographic preferred orientations [46,47]. To take this into account we combined the two phase growth code from the second project with a crystal plasticity FFT-based deformation code [48,49] in the third project.

The implementation is similar to Griera et al. [50] and is also described in chapter 5.2.2 in appendix 3 [34]. Adjustments to the code were necessary to make a seamless integration of both codes possible.

THE GROWTH PARAMETER FOR STATIC GRAIN GROWTH

The first publication [51] of this thesis deals with the issue of static grain growth in competition with other processes that modify the grain size, such as polygonisation [52-58]. The general growth law for static grain growth, which describes the increase in grain size solely driven by the reduction of free energy of the grain boundaries is [59-65]:

$$D_t^n - D_0^n = kt \qquad (1)$$

With D_t being the mean grain diameter at time t after grain growth stared at time $t=0$. The growth exponent n and the growth parameter k can be determined experimentally by sampling the microstructure at different times t after grain growth commenced. For the given process this growth law describes the grain-size evolution very well. However, in natural samples it is almost impossible to determine D_0 (which is a state parameter and not a material property) and a simplification of Eq. (1) is sometimes used, which is approximately valid when $D_0 << D_t$:

$$D_t^n \approx kt \ (if\ D_0 \ll D_t) \qquad (2)$$

Theory predicts that, for ideal static grain growth with isotropic grain-boundary energies, the growth exponent n equals 2 [62]. As soon as other processes or factors than reduction of free energy of the grain boundaries contribute to the grain growth, n increases [54,66-68]. Under perfect analytical conditions with only specific processes contributing to grain growth it can be shown that each process has a specific growth exponent n [64]. For example, Ostwald ripening with diffusion through the grain interiors should have an exponent of $n=3$ and diffusion along grain boundaries an exponent of $n=4$.

The parameter k is normally treated as a temperature dependent material property that is a function of the boundary energy γ and mobility M:

$$k = k_0 \gamma M \qquad (3)$$

The temperature dependence T of k can be expressed in relation to the activation energy Q and the universal gas constant R.

$$k \propto exp^{-Q/RT} \qquad (4)$$

It is often assumed that k_0 is a material-independent constant, which is 4.48 for grain growth in 2 dimensions and 2 in 3 dimensions [69]. In the first publication [51], however, we show that this parameter is not a constant. It is dependent on microstructure and can even vary during the same experiment if the starting microstructure is not a perfect foam texture produced by static grain growth only but contains remnant internal energy and other influences from different processes [22,51]. This is of importance for various reasons. Hiraga [70,71], Montagnat [72], Mathiesen [73] combined the standard growth law (Eq. 2) with a constant k_0 with other processes that modify the grain size. This, however, leads to erroneous results if the effect of these processes on k_0 is ignored. Furthermore, wrong values of n and k are obtained if during experiments the microstructure changes. This typically leads to an overestimate of n, which in turn affects the inferred rate-controlling process.

INFLUENCE OF BUBBLES ON GRAIN GROWTH

It was found in the first chapter that the microstructure plays an important role in determining the grain growth rate. While that chapter dealt with a pure, single-phase material, the second chapter investigates the effect of impurities on grain growth [11,22,54,74,75].

A common "impurity" or second phase in natural ice, at least in the upper part of ice sheets, are air bubbles [20,76]. With increasing depth and compaction they eventually transform into clathrates. The transition zone lies approximately in a range between 600m and 1200m depth [21,77-79]. To include a second phase with different properties in the simulations required significant additions and modifications to the simulation code, which are described in Appendix 4.

In a two-phase granular material (phases A and B) there are three types of boundaries: A-A, A-B and B-B boundaries. The number of boundary types increases for more phases. However, considering the focus on ice in this thesis, the numerical simulations are limited to two phases: ice and air. In this case only two types of boundaries exist in nature: those between ice and

ice, and between ice and air. For numerical reasons, the simulation code actually also allows virtual air-air boundaries.

The main additional complexity, compared to grain growth in a single-phase aggregate, is to maintain a mass balance: the fraction of each phase should remain constant. In an earlier version of the code, developed by Becker et al. [45,80], movement of boundaries maintained an overall mass balance, which implies that regions of each phase are fully connected throughout the model. Where this may be permissible for second-phase regions with a low dihedral angle (partially molten rocks), it is not for unconnected air bubbles in ice. This required a major revision of the code to ensure that the area of each individual bubble is maintained in the 2D model.

In general impurities or additional phases in a grain aggregate complicate the growth process from single phase materials quite significantly [81]. Second-phase particles may stop or hinder the movement of grain boundaries. This effect can be described by Zener pinning [82-84] and if the particles completely stop boundary movement they come to a complete stop when all boundaries are pinned [81,85].

Instead of parameterising the effect of air bubbles in modelling grain growth, air bubble boundaries were assigned different properties from those of ice-ice boundaries. For grain-boundary driven grain growth in the ice-air aggregate, the following parameters need to be known: surface energies of ice-ice and ice-air boundaries (γ_{ii} and γ_{ia}), as well as their mobilities (M_{ii} and M_{ia}). γ_{ii} is reasonably well constrained from experiments [86]. γ_{ia} is less well known, but can be inferred from the approximately spherical shape of air bubbles, which implies a dihedral angle (ω) that is close to 180°. Knowing γ_{ii}, one can determine γ_{ia}, using:

$$\omega = 2 \cdot cos^{-1}\left(\frac{\gamma_{ii}}{2 \cdot \gamma_{ia}}\right) \qquad (5)$$

In the simulations it was assumed that $\gamma_{ia} = 8 \cdot \gamma_{ii}$, which corresponds to ω=174°.

The only variable remaining in the equations is therefore the boundary mobility M. We decided to vary it by ratios between mobility for ice-air boundaries compared to mobility for ice-ice boundaries instead of absolute values. That means a ratio of R = 10 defines a setting where the air-ice boundaries have a mobility which is one order of magnitude higher than the

mobility setting for ice-ice boundaries. The third type of boundary, the air-air boundaries were introduced for numerical reasons, but they were assigned surface energy and mobility values such that the dummy air-air boundaries did not affect the behaviour of the system.

The results of this work generally confirm and further develop ideas from the first publication [51]. The growth parameter *k* is not a constant as is often assumed. It is a function of microstructure and varies with it. Three different growth regimes could be recognized in the simulations of ice-air aggregates. These depend on how many bubbles or second phase inclusions are in the microstructure and how they are distributed. If in the beginning there are many bubble-free boundaries, these can still migrate relatively unhindered and the resulting growth rate is relatively fast. The growth parameter *k* is initially close to that for pure ice but drops quickly once the bubble spacing gets close to the grain size. Since everything slows down the decline of *k* does too. In the final regime, a steady state is reached where the microstructure and topology do not change, except for an increase in both grain and bubble size. In this regime the low *k* remains approximately constant [74].

IMPACT OF DEFORMATION ON GRAIN BOUNDARY MIGRATION

So far only static grain growth and an abstract polygonisation parameter, expressed as chance for each grain to split in two each step was included in the experiments. Investigation on how static grain growth rates change with microstructure and when a different phase is added followed. However the "internal energy" was only mentioned as a factor which can also influence microstructure and a splitting parameter to simulate that to a certain extent was used [51]. In the third project, a more detailed simulation of dynamic recrystallization driven by this internal energy or dislocation density, and the formation of new grain boundaries by sub-grain rotation [55] was carried out. An implementation [87] of the full stress/strain field solving crystal plastic code using Fast Fourier Transformation [48,49] was incorporated in the experiments. It is described in chapter 5.2.2. in appendix 3 [34]. Since the main work is not yet ready to be published, a more detailed description follows. As a first step the experiments were restricted to single-phase pure ice simulations again. An adjustment in the code to enable two-phase simulations with air bubbles was made and tested successfully later. However a simple setting was chosen to ensure that the experiments were working correctly.

A detailed description of the underlying process can be found in Lebensohn et al. [48,49] and Griera et al. [87]. Their models were implemented for hexagonal ice 1h, which deforms by slip on the basal, prismatic and pyramidal planes. It was assumed that the basal plane is the easy-slip plane with a critical resolved shear stress 20 times lower than that for the other slip systems [20]. Chapter 5.2 in appendix 3 [34] gives a detailed description of how the various processes involved, while only a short description is given in the following.

The crystal structure and the orientation of the individual crystals are described in the Elle format [41,45,68] with grains defined by polygons (flynns). Additional to the previous experiments there is a regularly spaced square grid (u-nodes) on top of this structure. This is necessary since the FFT method only works on grids with n^{th} power of two elements along its sides. At the beginning the lattice-orientation information is copied to this grid which is then deformed by the given parameters. The FFT formulation provides an exact solution for the stress and strain field, including rotation of the crystal lattice within the individual elements [48,49]. From the angular mismatches of the lattice in adjacent elements, the least number of geometrically necessary dislocations with the lowest internal energy is calculated. After that recovery processes, such as tilt wall formation or annihilation of dislocations [88] (implementation after Borthwick, V. – unpublished) is activated. Deformation and recovery are calculated based on the regular u-node grid. Deformation is also applied to grain boundaries by moving their b-nodes according to the local velocity field. As the FFT-routine requires a regular grid, the deformed grid is mapped back onto the regular grid for the next FFT calculation.

Nucleation of new grains is carried out by creating new flynns at sites of high local misorientations, which is equivalent to high internal strain. The small new grains are given a random lattice orientation. The final process in the loop is grain-boundary migration. Additional to what was described in the first two publications, this process now additionally takes strain energy energy into account as well to determine the movement direction and magnitude of the individual boundary nodes. It gets the internal energy from the dislocation density map stored in the finite element grid. High dislocation density means high internal energy and the boundaries most likely will move in that direction to erase this high energy field. Once overgrown or recrystallized the dislocation density is reset to zero and the orientations of these grid points are adjusted to that of the neighbouring points inside the mother grain.

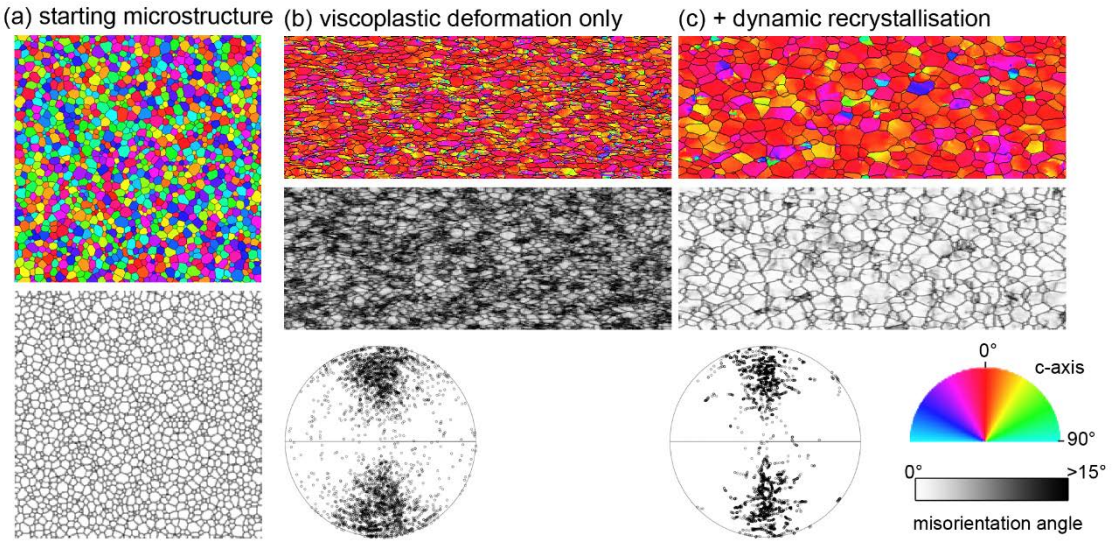


FIGURE 1 AFTER FIG 24 IN APPENDIX 3 [34]. FFT VISCOPLASTIC SIMULATION OF THE DEFORMATION OF A PURE ICE AGGREGATE WITH AND WITHOUT DYNAMIC RECRYSTALLISATION (RECOVERY AND GRAIN-BOUNDARY MIGRATION). TOP ROW SHOWS C-AXIS ORIENTATUNION, LOWER ROW SHOWS LOCAL MISORIENTATION AND TWO C-AXIS DISTRIBUTION PATTERNS. (A) SHOWS THE STARTING MICROSTRUCTURE WITH A RANDOM LATTICE ORIENTATION FOR EACH GRAIN. (B) SHOWS THE RESULTS AFTER 40% SHORTENING WITH VISCOPLASTIC DEFORMATION ONLY. (C) SHOWS THE RESULTS WITH THE SAME SETTINGS BUT ENABLED RECRYSTALLIZATION PROCESSES.

Figure 1 summarizes the work that is also described in appendix 3 [34]. Starting from a foam texture with randomly oriented lattice orientations several experiments simulated how adding different recovery processes influences the final microstructure. Viscoplastic deformation only results in elongated grains with very unrealistic boundary shapes and very high dislocation densities in regions of strain localisation. This can be seen best in the local misorientation map. A misorientation of more than 15° from the neighbouring element results in a black line. The high amount of black in the local misorientation map shows that after 40% shortening, almost every element in the model has a high-angle boundary with its neighbours, especially where strain is localisatized. Adding grain boundary migration and recovery to the system lowers the internal misorientations and adds moving boundaries and growing grains to the experiment. The unrealistic boundaries from the pure viscoplastic experiment are now replaced by smoothly curved boundaries. Grains with high internal energies (lattice orientations at an unfavourable orientation relative to the deformation field) are preferentially consumed by neighbouring grains. As a result the c-axis pattern in Figure 1 shows a more focused maxima. The work started in this PhD thesis will be continued in at least one more

publication with focus on a more detailed description and discussion of findings in our latest work.

REFERENCES

1 Zachos, J., Pagani, M., Sloan, L., Thomas, E. & Billups, K. Trends, Rhythms, and Aberrations in Global Climate 65 Ma to Present. *Science* **292**, 686-693 (2001).

2 Royer, D. L., Berner, R. A., Montañez, I. P., Tabor, N. J. & Beerling, D. J. CO2 as a primary driver of Phanerozoic climate. *GSA Today* **14**, 4-10, doi:10.1130/1052-5173(2004)014<4:CAAPDO>2.0.CO;2 (2004).

3 Peizhen, Z., Molnar, P. & Downs, W. R. Increased sedimentation rates and grain sizes 2±4 Myr ago due to the in¯uence of climate change on erosion rates. *Nature* **410**, 891-897 (2001).

4 Schwab, F. L. Modern and ancient sedimentary basins: Comparative accumulation rates. *Geology* **4**, 723-727, doi:10.1130/0091-7613(1976)4<723:MAASBC>2.0.CO;2 (1976).

5 Larsen, H. C. *et al.* Seven Million Years of Glaciation in Greenland. *Science* **264**, 952-955 (1994).

6 Hoffman, P. F. & Schrag, D. P. The snowball Earth hypothesis: testing the limits of global change. *Terra Nova* **14**, 129-155 (2002).

7 Kirschvink, J. L. *et al.* Paleoproterozoic snowball Earth: Extreme climatic and geochemical global change and its biological consequences. *PNAS* **97**, 1400-1405 (2000).

8 Hoffman, P. F., Kaufman, A. J., Halverson, G. P. & Schrag, D. P. A Neoproterozoic Snowball Earth. *Science* **281**, 1342-1346 (1998).

9 Ekaykin, A. A. *et al.* The changes in isotope composition and accumulation of snow at Vostok station, East Antarctica, over the past 200 years. *Annals of Glaciology* **39**, 569-575 (2004).

10 Frezzotti, M. *et al.* New estimations of precipitation and surface sublimation in East Antarctica from snow accumulation measurements. *Climate Dynamics* **23**, 803-813, doi:10.1007/s00382-004-0462-5 (2004).

11 Faria, S. H., Freitag, J. & Kipfstuhl, S. Polar ice structure and the integrity of ice-core paleoclimate records. *Quaternary Science Reviews* **29**, 338-351 (2010).

12 Petit, J. R. *et al.* Climate and atmospheric history of the past 420,000 years from the Vostok ice core, Antarctica. *Nature* **399**, 429-436 (1999).

13 Members, E. C. Eight glacial cycles from an Antarctic ice core. *Nature* **429**, 623–628, doi:10.1038/nature02599 (2004).

14 Jouzel, J. *et al.* Orbital and millennial Antarctic climate variability over the past 800 000 years. *Science* **317**, 793-796, doi:10.1126/science.1141038 (2007).

15 Solomon, S., Plattner, G.-K., Knutti, R. & Friedlingstein, P. Irreversible climate change due to carbon dioxide emissions. *PNAS* **106**, 1704-1709, doi:10.1073/pnas.0812721106 (2009).

16 Huss, M., Usselmann, S., Farinotti, D. & Bauder, A. Glacier mass balance in the south-eastern Swiss Alps since 1900 and perspectives for the future. *Erdkunde* **64**, 119-140 (2010).

17 Pelto, M. S. Forecasting temperate alpine glacier survival from accumulation zone observations. *The Cryosphere* **4**, 67-75 (2010).

18 Kaplan, M. R. *et al.* Glacier retreat in New Zealand during the Younger Dryas stadial. *Nature* **467**, 194-197, doi:doi:10.1038/nature09313 (2010).

19 Petrenko, V. F. & Whitworth, R. W. *Physics of Ice*. (Oxford University Press, 1999).

20 Schulson, E. M. & Duval, P. *Creep and Fracture of Ice*. (Cambridge University Press, 2009).

21 Hondoh, T. in *Physics of Ice Core Records II* Vol. 68 (ed T. Hondoh) 1-23 (Hokkaido University Press, 2009).

22 Arena, L., Nasello, O. B. & Levi, L. Effect of Bubbles on Grain Growth in Ice. *J. Phys. Chem. B* **101**, 6109-6112 (1997).

23 Dunaeva, A. N., Antsyshkin, D. V. & Kuskov, O. L. Phase diagram of H2O: Thermodynamic functions of the phase transitions of high-pressure ices. *Solar System Research* **44**, 202-222 (2010).

24 Patchett, P. J. Thermal effects of basalt on continental crust and crustal contamination of magmas. *Nature* **283**, 559-561 (1980).

25 Joughin, I., Smith, B. E., Howat, I. M., Scambos, T. & Moon, T. Greenland flow variability from ice-sheet-wide velocity mapping. *J Glaciol* **56**, 415-430 (2010).

26 Seddik, H., Greve, R., Zwinger, T. & Placidi, L. A full stokes ice flow model for the vicinity of Dome Fuji, Antarctica, with induced anisotropy and fabric evolution. *The Cryosphere* **5**, 495-508 (2011).

27 Seddik, H., Greve, R., Zwinger, T., Gillet-Chaulet, F. & Gagliardini, O. Simulations of the Greenland ice sheet 100 years into the future with the full Stokes model Elmer/Ice. *J Glaciol* **58**, 427-440 (2012).

28 Murray, J. Volcanic ashes and cosmic dust. *Nature*, 585-590 (1884).

29 Carey, S. & Sparks, R. S. J. Quantitative models of the fallout and dispersal of tephra from volcanic eruption columns. *Bulletin of Vulcanology* **48**, 109-125 (1986).

30 Gow, A. J. & Williamson, T. Volcanic ash in the Antarctic ice sheet and its possible climate implications. *Earth Planet Sc Lett* **13**, 210-218 (1971).

31 Narcisi, B., Petit, J. R., Delmonte, B., Basile-Doelsch, I. & Maggi, V. Characteristics and sources of tephra layers in the EPICA-Dome C ice record (East Antarctica): Implications for past atmospheric circulation and ice core stratigraphic correlations. *Earth Planet Sc Lett* **239**, 253-265 (2005).

32 Mortensen, A. K., Bigler, M., Grönvold, K., Steffensen, J. P. & Johnsen, S. J. Volcanic ash layers from the Last Glacial Termination in the NGRIP ice core. *Journal of Quaternary Science* **20**, 209-219 (2005).

33 Seward, D. & Kohn, B. P. New zircon fission-track ages from New Zealand Quaternary tephra: an interlaboratory experiment and recommendations for the determination of young ages. *Chemical Geology* **141**, 127-140 (1997).

34 Montagnat, M. *et al. Journal of Structural Geology* (submitted).

35 Castelnau, O., Brenner, R. & Lebensohn, R. A. The effect of strain heterogeneity on the work-hardening of polycrystals predicted by mean-field approaches. *Acta Materialia* **54**, 2745-2756 (2006).

36 Castelnau, O., Canova, G. R., Lebensohn, R. A. & Duval, P. Modelling viscoplastic behavior of anisotropic polycrystalline ice with a self-consistent approach. *Acta Materialia* **45**, 4823-4834 (1997).

37 Faria, S. H. Creep and recrystallization of large polycrystalline masses. I. General continuum theory. *Proceedings of the Royal Society A* **462**, 1493-1514, doi:10.1098/rspa.2005.1610 (2006).

38 Faria, S. H. Creep and recrystallization of large polycrystalline masses. III. Continuum theory of ice sheets. *Proceedings of the Royal Society A* **462**, 2797-2816, doi:10.1098/rspa.2006.1698 (2006).

39 Faria, S. H., Kremer, G. M. & Hutter, K. Creep and recrystallization of large polycrystalline masses. II. Constitutive theory for crystalline media with transversely isotropic grains. *Proceedings of the Royal Society A* **462**, 1699-1720, doi:10.1098/rspa.2005.1635 (2006).

40 Jessell, M., Bons, P., Evans, L., Barr, T. & Stuwe, K. Elle: the numerical simulation of metamorphic and deformation microstructures. *Computers & Geosciences* **27**, 17-30 (2001).

41 Bons, P. D., Koehn, D. & Jessell, M. W. *Microdynamic Simulation*. Vol. 106 405 pp (Springer, 2008).

42 Piazolo, S., Jessell, M. W., Bons, P. D., Evans, L. & Becker, J. K. Numerical Simulations of Microstructures Using the Elle Platform: A Modern Research and Teaching Tool. *Journal of the Geological Society of India* **75**, 110-127 (2010).

43 Jessell, M. W. & Bons, P. D. The numerical simulation of microstructure. *Geological Society, London, Special Publications* **200**, 137-147 (2002).

44 Jessell, M. W., Kostenko, O. & Jamtveit, B. The preservation potential of microstructures during static grain growth. *Journal of Metamorphic Geology* **21**, 481–491, doi:doi:10.1046/j.1525-1314.2003.00455.x. (2003).

45 Becker, J. K., Bons, P. D. & Jessell, M. W. A new front-tracking method to model anisotropic grain and phase boundary motion in rocks. *Computers & Geosciences* **34**, 201-212 (2008).

46 Jessell, M. W. Simulation of fabric development in recrystallizing aggregates. 1. Description of the models. *Journal of Structural Geology* **10**, 771-778 (1988).

47 Jessell, M. W. Simulation of fabric development in recrystallizing aggregates. 2. Example model runs. *Journal of Structural Geology* **10**, 779-793 (1988).

48 Lebensohn, R. A. N-site modeling of a 3D viscoplastic polycrystal using Fast Fourier Transform. *Acta Materialia* **49**, 2723-2737 (2001).

49 Lebensohn, R. A., Brenner, R., Castelnau, O. & Rollett, A. D. Orientation image-based micromechanical modelling of subgrain texture evolution in polycrystalline copper. *Acta Materialia* **56**, 3914–3926, doi:10.1016/j.actamat.2008.04.016 (2008).

50 Griera, A. *et al.* Numerical modelling of porphyroclast and porphyroblast rotation in anisotropic rocks. *Tectonophysics*, in press (2011).

51 Roessiger, J. *et al.* Competition between grain growth and grain-size reduction in polar ice. *J Glaciol* **57**, 942-948 (2011).

52 Alley, R. B. Flow-law hypotheses for ice-sheet modelling. *J Glaciol* **38**, 245-256 (1992).

53 De La Chapelle, S., Castelnau, O., Lipenkov, V. & Duval, P. Dynamic recrystallization and texture development in ice as revealed by the study of deep ice cores in Antarctica and Greenland. *Journal of Geophysical Research* **103**, 5091-5105 (1998).

54 Durand, G. *et al.* Effect of impurities on grain growth in cold ice sheets. *Journal of Geophysical Research* **111**, F01015 (2006).

55 Urai, J. L., Means, W. D. & Lister, G. S. in *Mineral and Rock Deformation: Laboratory Studies* Vol. 36 (eds B. E. Hobbs & H. C. Heard) 161-200 (Geophysical Monograph, 1986).

56 Alley, R. B., Gow, A. J. & Meese, D. A. Mapping c-axis fabrics to study physical processes in ice. *J Glaciol* **41**, 197-203 (1995).

57 Duval, P. & Castelnau, O. Dynamic recrystallisation of ice in polar ice sheets. *J Phys-Paris* **5**, 197-205 (1995).

58 Faria, S. H., Ktitarev, D. & Hutter, K. Modelling evolution of anisotropy in fabric and texture of polar ice. *Annals of Glaciology* **35**, 545-551 (2002).

59 Alley, R. B., Perepezko, J. H. & Bentley, C. R. Grain Growth in polar ice: I. Theory. *J Glaciol* **32**, 415-424 (1986).

60 Smith, C. S. Some elementary principles of polycrystalline microstructure. *Metallurgical Reviews* **9**, 1-48 (1964).

61 Weaire, D. & Rivier, N. Soap, cells and statistics - random patterns in two dimensions (Reprinted from Contemporary Physics, vol 25, pg 59, 1984). *Contemporary Physics* **50**, 199-239 (2009).

62 Glazier, J. A., Gross, S. P. & Stavans, J. Dynamics of Two-Dimensional Soap Froths. *Phys Rev A* **36**, 306-312 (1987).

63 Anderson, M. P. in *Annealing processes - Recovery, Recrystallization and grain growth* (eds N. Hanse, N. Juul Jensen, D. Leffers, & T. B. Ralp) 15-34 (Risø National Laboratory, 1986).

64 Evans, B., Renner, J. & Hirth, G. A few remarks on the kinetics of static grain growth in rocks. *Int J Earth Sciences* **90**, 88-103, doi:DOI 10.1007/s005310000150 (2001).

65 Weygand, D., Bréchet, Y., Lépinoux, J. & Gust, W. Three dimensional grain growth: a vertex dynamics simulation. *Philos. Mag. B,* **79**, 703-716 (1998).

66 Gow, A. J. On the rates of growth of grains and crystals in south polar firn *J Glaciol* **8**, 241-252 (1969).

67 Gow, A. J. *et al.* Physical and structural properties of the Greenland Ice Sheet Project 2 ice core: A review. *Journal of Geophysical Research* **102**, 26559-26575 (1997).

68 Bons, P. D., Jessell, M. W., Evans, L., D., B. T. & K., S. Modelling of anisotropic grain growth in minerals. *Geological Society of America Memoir* **193**, 39-49 (2001).

69 Mullins, W. W. Estimation of the Geometrical Rate-Constant in Idealized 3 Dimensional Grain-Growth. *Acta Metallurgica* **37**, 2979-2984 (1989).

70 Hiraga, T., Miyazaki, T., Tasaka, M. & Yoshida, H. Mantle superplasticity and its self-made demise. *Nature* **468**, 1091–1094, doi:doi:10.1038/nature09685 (2010).

71 Hiraga, T., Tachibana, C., Ohashi, H. & Sano, S. Grain growth systematics for forsterite ± enstatite aggregates: Effect of lithology on grain size in the upper mantle. *EPSL* **291**, 10-20 (2010).

72 Montagnat, M. & Duval, P. Rate controlling processes in the creep of polar ice, influence of grain boundary migration associated with recrystallization. *EPSL* **183**, 179-186 (2000).

73 Mathiesen, J. *et al.* Dynamics of crystal formation in the Greenland NorthGRIP ice core. *J Glaciol* **50**, 325-328 (2004).

74 Roessiger, J., Bons, P. D. & Faria, S. H. Influence of bubbles on grain growth in ice. *Journal of Structural Geology* **in press**, doi:http://dx.doi.org/10.1016/j.jsg.2012.11.003 (2012).

75 Azuma, N., Miyakoshi, T., Yokoyama, S. & Takata, M. Impeding effect of air bubbles on normal grain growth of ice. *Journal of Structural Geology*, doi:http://dx.doi.org/10.1016/j.jsg.2012.05.005 (2012).

76 Arnaud, L., Barnola, J. M. & Duval, P. in *Physics of Ice Core Records* (ed T. Hondoh) 285-305 (Hokkaido University Press, 2000).

77 Barnes, P. R. F., Mulvaney, R., Robinson, K. & Wolff, E. W. Observations of polar ice from the Holocene and the

glacial period using the scanning electron microscope. *Annals of Glaciology* **35**, 559-566 (2002).
78 Lipenkov, V. Y., Salamati, A. N. & Duval, P. Bubbly-ice densification in ice sheets: II. Applications. *J Glaciol* **43**, 398-407 (1992).
79 Faria, S. H. *et al.* in *Physics of Ice Core Records II* (ed T. Hondoh) 39-59 (Hokkaido University Press, 2009).
80 J.K. Becker, N. P. W., M. Jessel, P.D. Bons, C.W. Passchier, L. Evans. Numerical simulation of disequilibrium structures in solid-melt systems during grain growth. *J. Virtual Explor.* **11** (2003).
81 Herwegh, M., Linckens, J., Ebert, A., Berger, A. & Brodhag, S. H. The role of second phases for controlling microstructural evolution in polymineralic rocks: a review. *Journal of Structural Geology* **33**, 1728-1750 (2011).
82 Olgaard, D. L. & Evans, B. Grain growth in synthetic marbles with added mica and water. *Contrib Mineral Petrol* **100**, 246-260 (1988).
83 Olgaard, D. L. & Evans, B. Effect of second-phase particles on grain growth in calcite. *Journal of the American Ceramic Society* **69**, C272-C277 (1986).
84 Brodhag, S. H. & Herwegh, M. The effect of different second-phase particle regimes on grain growth in two-phase aggregates: insights from in situ rock analogue experiments. *Contributions to Mineralogy and Petrology* **160**, 219-238 (2010).
85 Weygand, D., Bréchet, Y. & Lépinoux, J. Zener Pinning and Grain Growth: a two-dimensional vertex computer simulation. *Acta mater.* **47**, 961-970 (1999).
86 Ketcham, W. M. & Hobbs, P. V. An experimental determination of the surface energies of ice. *Philosophical Magazine* **19**, 1161-1173 (1969).
87 Griera, A. *et al.* Strain localization and porphyroclast rotation. *Geology*, 275-278 (2011).
88 Passchier, C. W. & Trouw, R. A. J. *Microtectonics*. (Springer, 2005).

ACKNOWLEDGEMENTS

First of all I would like to thank my supervisor Prof. Paul D. Bons. He always listened to problems during the long path from the beginning to the end. He also never gave up on trying to change my writing skills away from the narrative novel writing to a more scientific, precise and condensed writing. I didn't live up to this, somehow my mind can't cope with this style and always tries to fill in unnecessary words and personal feelings or emotions which most possibly aren't true for everyone who reads the work. I guess in the end I'm not capable to continue the scientific career. Thank you for trying Paul! I had a wonderful time in you working group, it was a special environment. And when it came to meeting deadlines it never felt like pressure and was always enjoyable. I really liked coming to work each day. Of course that also meant that some days were not very productive because I sometimes have a hard time to do something useful and end up with a day spent on unimportant things. Sometimes I was also invited to join him on field trips with students as a second supervisor and these days were always a great chance to see some outcrops again. My second supervisor Prof. Sérgio H. Faria also had patience with me and we had some very helpful discussions during the last years. His understanding for Physics astonished me many times, and I also enjoyed when our discussions on the phone and on workshops reached a more relaxed point after the technical part was done.

About ten month after I started my PhD my colleague Anett Weisheit started her PhD in the same working group and I couldn't think of a more helpful person than her. She was always open for a talk about everything. We had a lot of fun in our office and altogether she spent a lot of days helping me with all sorts of things. I guess she wasn't born with the ability to say no, a major disadvantage which I sometimes exploited. However I guess inwardly she knew and did it anyway. Otherwise it wouldn't have happened because she is a pretty clever person. When it came to technical bullshit I had to look for someone else to talk to. Balint Morvai happily filled that position and we spent a lot of hours together talking about computer stuff. Of course this wasn't the only topic which filled our Mensa lunch or coffee times. Balint knows something about almost everything and I'm still puzzled on how he manages to keep being up to date on so many topics. I'm happy that I met him along with all the other students in our working group. Jürgen, Philipp and also Simon were often available for long minutes of

babbling if necessary. Of course there were also a few people outside our working group who made the life as a PhD easier. Especially Wolfgang Siebel and Horst Hann always liked to listen to my new problems and also provided many tips for a future academic career and also for other things in life.

At home I also was always welcome and especially my mother Karin showed interest in my work. Regrettably I rarely showed the patience to explain all the scientific work to her. Most of the time I just said that simple translation wouldn't help to make her understand. Since she is not at all interested in computers it was also difficult to think of a way to make my work sound interesting for her.

Last but not least I would like to thank the DFG for the funding of my research project 1776/7 and all the interesting conference participations it made possible!

CONTRIBUTIONS

FIRST PAPER

TITLE: "COMPETITION BETWEEN GRAIN GROWTH AND GRAIN-SIZE REDUCTION IN POLAR ICE"

Idea and concept 40%. Code development, simulations and data collection 100%. Analysis and Interpretation 80%. Writing of paper 60%.

SECOND PAPER:

TITLE: "INFLUENCE OF BUBBLES ON GRAIN GROWTH IN ICE"

Idea and concept 60%. Code development, simulations and data collection 100%. Analysis and Interpretation 90%. Writing of paper 70%.

THIRD PAPER

TITLE: "MULTISCALE MODELLING OF ICE DEFORMATION BEHAVIOUR"

Idea and concept 8%. Code development, simulations and data collection 8%. Analysis and Interpretation 6%. Writing of paper 5%.

CODE

All the code described in this part of the appendix was written by myself. In part this involved modification and further development of existing algorithms and code (marked with [a] below), while in other cases completely new code had to be written (marked with [b]).

- Different approach to the splitting process [(b)]
- Combination of growth & split [(a)]
- Two phase grain growth with different options to keep the area constant
    - First approach by geometrical movement restrictions [(b)]
    - Second approach by using boundary nodes directly [(b)]
    - Third approach by using grains and clusters of grains [(b)]

- Combination of two phase grain growth & the FFT deformation approach (together with Albert Griera). [(a)]
- Several small routines and scripts to simplify things. [(b)]

APPENDIX 1

COMPETITION BETWEEN GRAIN GROWTH AND GRAIN-SIZE REDUCTION IN POLAR ICE

Competition between grain growth and grain-size reduction in polar ice

Jens ROESSIGER,[1] Paul D. BONS,[1] Albert GRIERA,[2] Mark W. JESSELL,[3] Lynn EVANS,[4] Maurine MONTAGNAT,[5] Sepp KIPFSTUHL,[6] Sérgio H. FARIA,[7] Ilka WEIKUSAT[6]

[1]*Institut für Geowissenschaften, Eberhard Karls Universität, Wilhelmstrasse 56, D-72074 Tübingen, Germany*
E-mail: jens.roessiger@uni-tuebingen.de
[2]*Departament de Geologia, Universitat Autònoma de Barcelona, ES-08193 Bellaterra, Spain*
[3]*IRD LMTG UMR 5563, 14 avenue Edouard Belin, 31400 Toulouse Cedex, France*
[4]*School of Geosciences, Monash University, Clayton, Victoria 3800, Australia*
[5]*Laboratoire de Glaciologie et Géophysique de l'Environnement, CNRS/Université Joseph Fourier – Grenoble I, 54 rue Molière, BP 96, 38402 Saint-Martin-d'Hères Cedex, France*
[6]*Alfred Wegener Institute for Polar and Marine Research, Columbusstrasse, D-27568 Bremerhaven, Germany*
[7]*GZG, Department of Crystallography, University of Göttingen, Goldschmidtstrasse 1, D-37077 Göttingen, Germany*

ABSTRACT. Static (or 'normal') grain growth, i.e. grain boundary migration driven solely by grain boundary energy, is considered to be an important process in polar ice. Many ice-core studies report a continual increase in average grain size with depth in the upper hundreds of metres of ice sheets, while at deeper levels grain size appears to reach a steady state as a consequence of a balance between grain growth and grain-size reduction by dynamic recrystallization. The growth factor k in the normal grain growth law is important for any process where grain growth plays a role, and it is normally assumed to be a temperature-dependent material property. Here we show, using numerical simulations with the program Elle, that the factor k also incorporates the effect of the microstructure on grain growth. For example, a change in grain-size distribution from normal to log-normal in a thin section is found to correspond to an increase in k by a factor of 3.5.

INTRODUCTION

Many classical studies of polar ice microstructure report an evolution of the mean grain size with depth according to what can be called the 'three-stage model' (Gow and Williamson, 1976; Herron and Langway, 1982; Thorsteinsson and others, 1997): in the upper few hundred metres, grain size increases steadily with depth; below a certain intermediate depth (400–700 m), the grain size stabilizes and remains roughly constant; finally, at great depths (approximately the last 300 m before reaching bedrock, where temperature exceeds –10°C (De La Chapelle and others, 1998; Duval, 2000) the grain size significantly increases again. Here we only deal with the upper two regions, where grain size first increases and then stabilizes.

The initial steady increase in grain size is usually explained by *static ('normal') grain growth* (Smith, 1964; Alley and others, 1986; Weaire and Rivier, 2009), defined as growth that is only driven by the reduction of free energy of the grain boundaries. The increase in grain size, expressed in mean radius, r, from a starting grain size, r_0, is usually described by (Anderson, 1986; Glazier and others, 1987; Weygand and others, 1998)

$$r^n - r_0^n = kt. \quad (1)$$

The growth exponent n has a theoretical value of 2 in ideal static grain growth of grains with isotropic properties (Glazier and others, 1987). In natural systems, the exponent is usually found to be >2. Any other process or factor that influences grain growth tends to increase n, such as anisotropic boundary energies, pinning, etc. (Gow, 1969; Gow and others, 1997; Bons and others, 2001; Durand and others, 2006). The parameter k is normally treated as a temperature-dependent material property that is a function of only the boundary energy $\gamma(T)$ and the grain boundary mobility $M(T)$:

$$k = k_0 \gamma M, \quad (2)$$

where T is the temperature and the factor k_0 is generally assumed to be constant. For ideal static grain growth the value of k_0 is ~0.5 in three dimensions and ~1.12 in two dimensions (Mullins, 1989; Weygand and others, 1998). Below we show that in practice k_0 is actually not a constant, but in fact depends on the microstructure (the ideal case being a particular instance). The factor k_0 itself is usually difficult to determine from experiments or measurements in nature (i.e. polar ice caps). This is because one normally only obtains k, which also includes the surface energy and grain boundary mobility. If k depends on microstructure through the parameter k_0, one cannot apply k obtained from one study to another situation where the microstructure may be different. In this paper we show that k_0 varies with microstructure and how ignoring this may lead to erroneous results if applied to polar ice caps.

If static grain growth were the only process operating in polar ice, the grain size should increase steadily with the age of the ice, and hence with depth. The observation in several ice cores that grain size stabilizes at a certain depth suggests that another process operates which balances the increase in grain size (Alley, 1992; De La Chapelle and others, 1998; Durand and others, 2006). If this other process leads to a reduction of grain size, a balance between grain-size increase and decrease will be reached at some point. The process usually invoked to explain the grain-size reduction process is polygonization or rotational/continuous recrystallization (Urai and others, 1986; Alley, 1992; Alley and others, 1995; Duval and Castelnau, 1995; Faria and others, 2002).

Rotational recrystallization is a deformation-driven process. Deformation by dislocation creep introduces dislocations in the crystal lattice, which can accumulate in planar zones or tilt walls that define regions within a grain with small differences in their lattice orientations. The lattice within these regions or subgrains within a grain thus rotate relative to each other. Progressive rotation of the subgrains with ongoing strain eventually leads to the formation of high-angle grain boundaries, and the subgrains they bound become real grains (Read, 1953; Duval and others, 1983). Rotational recrystallization can be regarded as a process that effectively splits grains into two or more grains (Mathiesen and others, 2004; Placidi and others, 2004). Each split increases the number of grains, N, in a volume by 1. The increase in N, and hence decrease in grain size, thus depends on the split rate f per grain:

$$\frac{\mathrm{d}N}{\mathrm{d}t} = fN. \quad (3)$$

The parameter f may depend on many factors, most importantly on strain rate and hence on deviatoric stress (e.g. through Glenn's flow law; Alley, 1992). However, in a first approximation it is usually assumed that the strain rate is approximately constant within the upper part of the core where our calculations apply (Lipenkov and others, 1989; Thorsteinsson and others, 1997; Montagnat and Duval, 2000). The split rate of a grain probably also depends on the size and deformation history of that grain. A split rate proportional to grain size was, for example, assumed by Mathiesen and others (2004) and Placidi and others (2004), while Morland (2009) studied the effect of ice flow history. However, the simplest (but not necessarily realistic) assumption is that f is a constant, not depending on grain size or any other factor. This simplification is permissible here, since this paper is mainly concerned with the influence of microstructure on growth rate, and we do not intend to model a particular ice core. For this case, a simple analytical solution exists for the stable grain size. Assuming that the grain growth exponent n is 2 in Equation (1), one derives (see Appendix)

$$\frac{\mathrm{d}N_{(t)}}{\mathrm{d}t} = -\frac{3ka^{2/3}}{2}{N_{(t)}}^{5/3} + fN_{(t)} \Leftrightarrow {r_{(t)}}^2 = \frac{3k}{2f}\left(1 - \mathrm{e}^{\frac{-2ft}{3}}\right). \quad (4)$$

Here a is a geometrical factor relating the mean grain radius, r, to the number, N, of grains in a volume. For illustration, by applying this equation to the North Greenland Icecore Project (NorthGRIP) ice-core data (Fig. 1), one obtains a growth constant of $k \approx 5.0 \times 10^{-3}\,\mathrm{mm^2\,a^{-1}}$ and a split rate of $f \approx 1.5 \times 10^{-3}\,\mathrm{a^{-1}}$ or once every 650 years. These numbers are within the range of those reported in the literature (Gow, 1969; Thorsteinsson and others, 1997; Svensson and others, 2003; Mathiesen and others, 2004). The question, however, is whether the values obtained are realistic and meaningful.

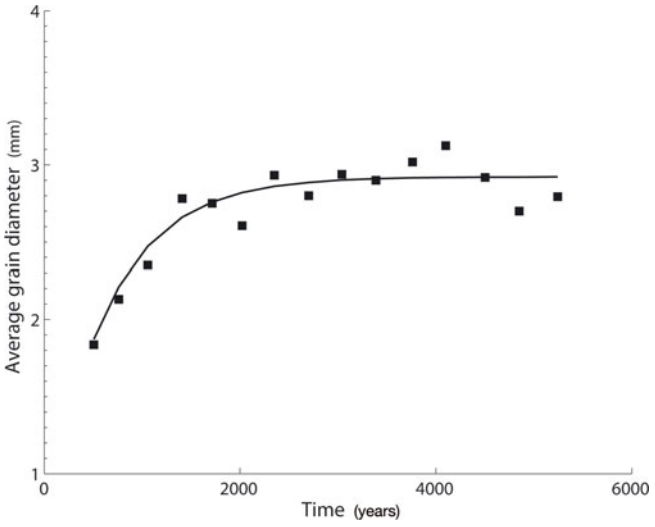


Fig. 1. Fit of analytical model (Equation (4)) to the average grain diameter as a function of age as observed in the NorthGRIP ice core (squares; data from fig. 3 in Mathiesen and others, 2004). Fit parameters are $k = 5.0 \times 10^{-3}\,\mathrm{mm^2\,a^{-1}}$ and a split rate of $f = 1.54 \times 10^{-3}\,\mathrm{a^{-1}}$ or once every 650 years.

NUMERICAL SIMULATIONS

We used the numerical modelling platform Elle (Jessell and others, 2001; Jessell and Bons, 2002; Bons and others, 2008) to simulate the process of grain growth and grain splitting. The Elle software was developed to simulate the microstructural evolution in materials such as rocks. It has been applied to the simulation of a range of processes, such as static grain growth in anisotropic polycrystals or partially molten rocks (Bons and others, 2001; Becker and others, 2008), dynamic recrystallization (Piazolo and others, 2002, 2004) and strain localization (Jessell and others, 2005). The main distinguishing features are (1) that it uses a two-dimensional (2-D) image of the actual microstructure, and (2) that it uses operator-splitting to allow a range of different processes to operate on, and modify the microstructure. This means that simultaneously operating processes (such as grain growth and grain splitting) are modelled as isolated individual processes that sequentially modify the microstructures in very small increments.

The microstructure is defined by a contiguous set of polygons that are themselves defined by boundary nodes that link straight boundary segments (Fig. 2). The polygons typically represent individual grains. Changes in the microstructure are achieved by (1) changing the properties of polygons or boundary nodes, (2) changing the position of boundary nodes, which implies a change in shape of the polygons, and (3) creating, removing or reordering boundary nodes and segments. A change in shape can be the result of deformation, for which the finite-element code, Basil, is available in Elle (Houseman and others, 2008). A change in shape can also be the result of the movement of boundaries (grain boundary migration), for example in the case of grain growth.

The movement of grain boundaries is modelled by sequentially selecting each boundary node, and applying a small incremental displacement that depends on the driving force for migration and the intrinsic boundary mobility. In this study we test the validity of Equation (4) by combining a static grain growth routine that moves grain boundaries, and a split routine that divides grains into two daughter grains.

The normal grain growth routine simulates ideal isotropic growth (without grain boundary energy anisotropy). For each time-step, the routine goes through the list of all boundary nodes and calculates the local radius of curvature, r_c, using the node and its immediate grain boundary neighbours. The velocity, v, of the node in the direction of the centre of the curvature is calculated using

$$v = \frac{M\gamma}{r_c} \text{ and } \Delta x = v \cdot \Delta t. \quad (5)$$

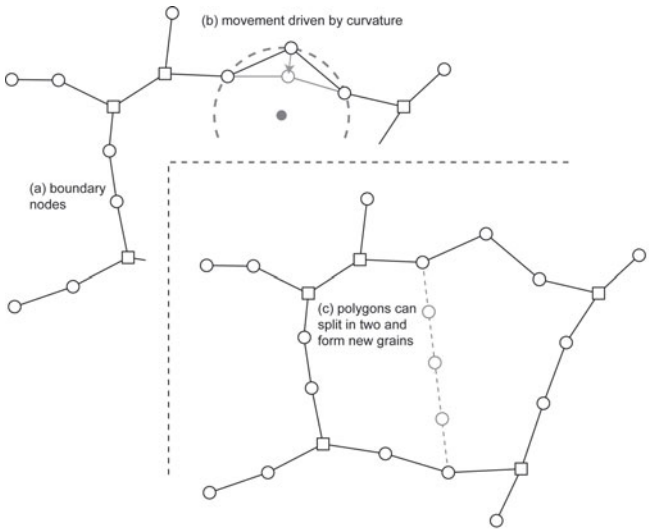


Fig. 2. Basic structure of the Elle model. The model consists of polygons which represent grains, and these polygons are in turn defined by boundary nodes (a) that are connected by straight boundary segments. Only boundary nodes with two or three neighbours are allowed in the model. The boundary nodes can move (b) and their movement is determined by the curvature of the boundary of the polygon at that point. Grains are split by the introduction of a new straight boundary that links two existing nodes (c).

The node is then moved over a distance Δx for a small time increment Δt. This routine results in ideal growth with a linear increase in mean grain area A, implying a growth exponent of $n=2$, and $k_0=1.22$ (Figs 3 and 4a). This would be the growth exponent as expected from theory (Humphreys and Hatherly, 1996). However, growth exponents measured in natural ice may deviate from that value due to other processes not taken into account here.

The effect of rotational recrystallization was implemented by randomly splitting each grain with a probability of f every time-step for each grain. This probability determines the rate of grain-size reduction by splitting. For this, each grain is selected in turn, and a random number generator determines whether the grain will be split. If so, one of its nodes is randomly selected and a new boundary is constructed across the grain, in a random orientation. Each time, the program checks whether the intended split will cause topological problems, such as intersection of the new boundary with another boundary or that a tiny grain has insufficient available nodes to split between. As a result, some splits are cancelled and a set value of f of $1.54\times10^{-3}\,\mathrm{a}^{-1}$ results in an effective split rate of $1.52\times10^{-3}\,\mathrm{a}^{-1}$, meaning that on average 1.3% of attempted splits are cancelled when a steady state has been established.

As expected, a stable grain size is established as a result of the combination of growth and splitting (Figs 4b and 5). For $M\gamma=3.2\times10^{-3}\,\mathrm{mm}^2\,\mathrm{a}^{-1}$ ($k=3.90\times10^{-3}\,\mathrm{mm}^2\,\mathrm{a}^{-1}$) and $f=1.52\times10^{-3}\,\mathrm{a}^{-1}$), the average stable grain diameter is $3\,\mathrm{mm}^2$. To compare this result with the analytical model, we must rewrite Equation (4) for the 2-D case:

$$A=\frac{k}{f}\left(1-\mathrm{e}^{-ft}\right)\Rightarrow A_{t\to\infty}=\frac{k}{f}. \qquad (6)$$

The average stable grain area predicted by the analytical model (Equation (6)) is similar to the value obtained with the

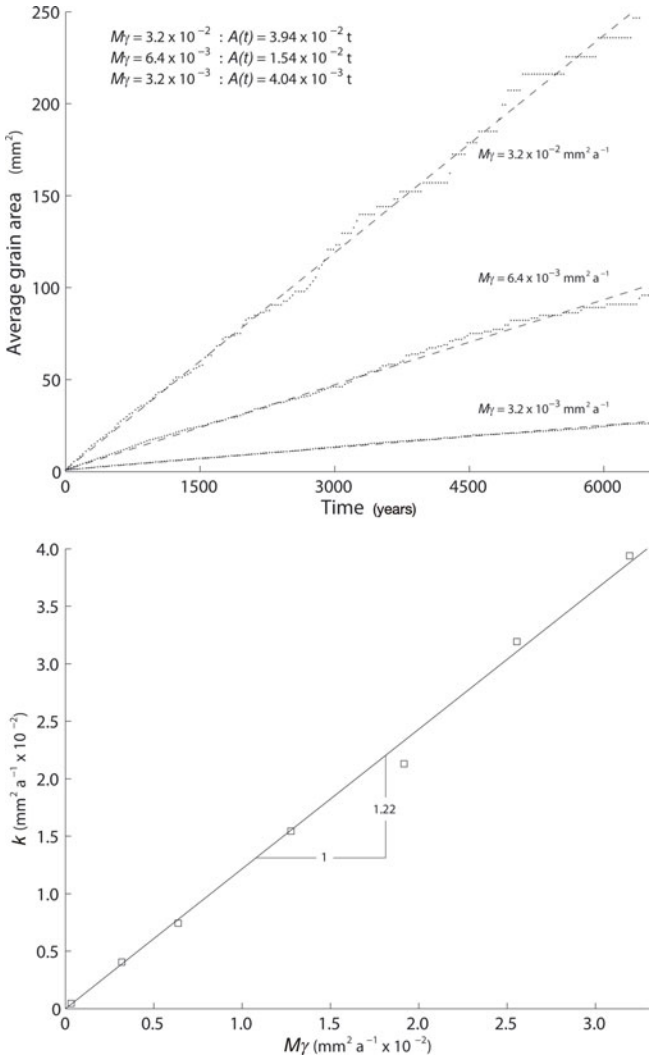


Fig. 3. (a) Growth curves for models of pure static grain growth. For $M\gamma=3.2\times10^{-3}$, 6.4×10^{-3} and $3.2\times10^{-2}\,\mathrm{mm}^2\,\mathrm{a}^{-1}$, the average grain area increases linearly with time. (b) Plot of k values measured from simulations as a function of the set value of $M\gamma$. The slope of 1.22 is the value of k_0.

Elle simulation, although the stable state is only reached after ~4000 years in the simulation. To achieve stabilization of the grain size after ~2000 years, as in the case of the NorthGRIP data, one has to roughly double both k and f. The discrepancy between the analytical model (Equation (4)) and the numerical simulation can be explained by considering the microstructure (Fig. 4). Static grain growth produces a regular foam texture. The frequency distribution of grain diameter has a maximum at about the average grain area (Fig. 6), and the normalized grain-size distribution is time-invariant (for steady-state growth). When a stable grain size is reached due to a balance between grain boundary migration and splitting, the grain size distribution changes significantly, with an increase of the frequency of very small grains, but also an increase in grains much larger than the average.

The change in microstructure changes the growth behaviour. The relatively abundant small grains have a high boundary curvature and quickly disappear. Yet many new small grains constantly appear because in the model every grain has the same chance of being split, independent of its size. The effect of the widening of the grain-size spectrum is an increase in the growth rate that balances the split rate in

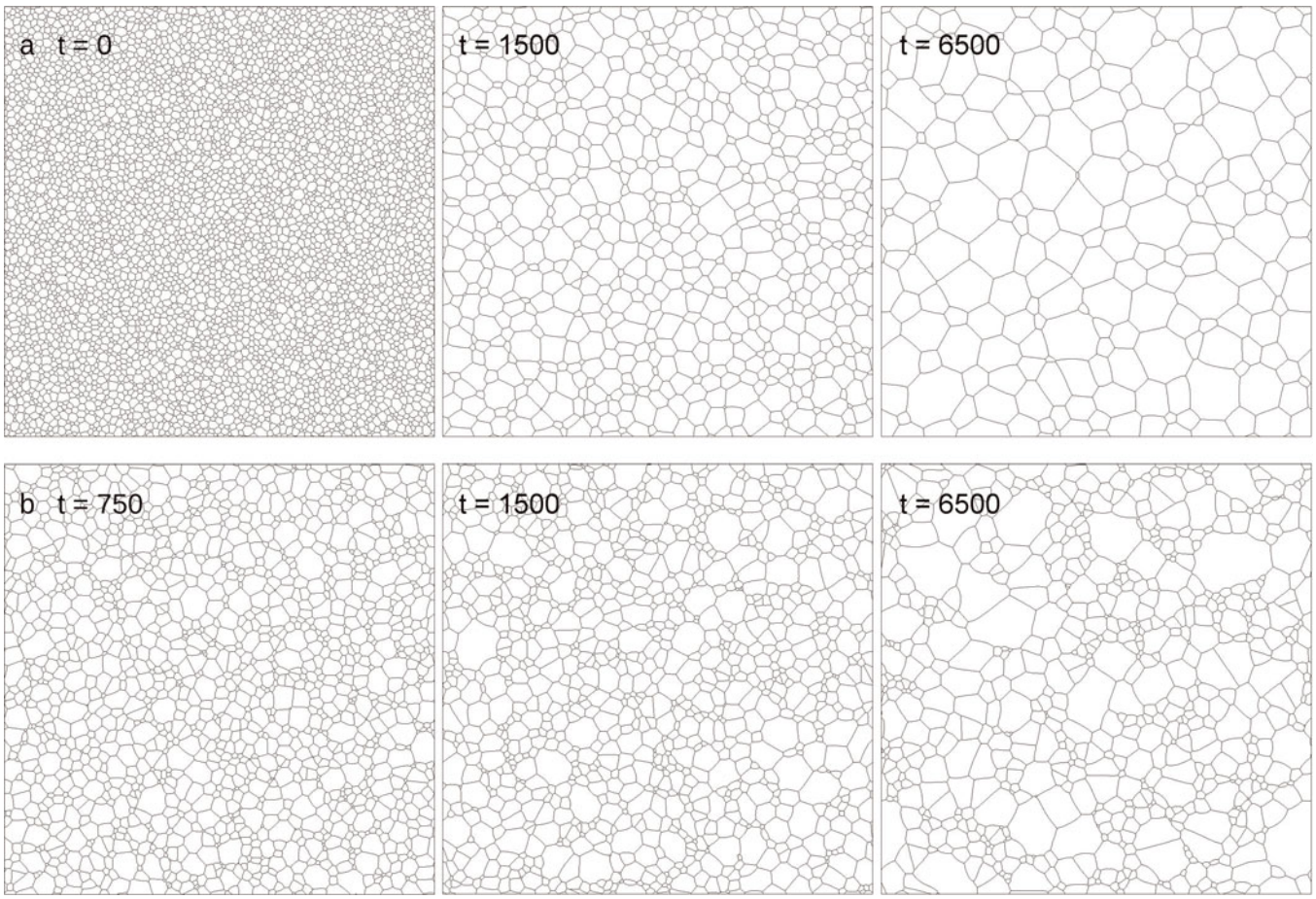


Fig. 4. Results of numerical simulations with Elle. (a) Static grain growth only, for 6500 years and $M\gamma = 3.2 \times 10^{-3}$ mm^2 a^{-1}. (b) Simulation with same starting aggregate and settings as for (a), but with splitting at a constant $f = 1.54 \times 10^{-3}$ a^{-1} added, which leads to the establishment of a stable grain size after ~4000 years, and a different microstructure compared to static grain growth. Size of box is 72 mm × 72 mm.

Equation (4). This can be seen if one stops the splitting when a stable grain size has settled but grain growth is allowed to continue (Fig. 7). The initial growth rate is over three times higher than the stable growth rate that is reached after the mean grain area has about quadrupled. This implies that the factor k_0 is not a constant, but a function of the microstructure. For the stable foam texture that results from static grain growth only, k_0 is 1.22. When the microstructure is the result of a competition and random splitting, the effective value of k_0 increases to 4.2 (an increase by a factor of 3.5).

DISCUSSION

The modelling in this paper is in no way intended to argue that the microstructure and grain size of the upper hundreds of metres of polar ice is determined by a balance of static grain growth and a constant grain-splitting rate. For this reason, we do not attempt to fit the results of the numerical simulations to obtain a growth constant or an average split rate of once every so many years. The dynamics of rotational recrystallization are much more complex (Faria and Kipfstuhl, 2004; Weikusat and others, 2011) than can be grasped by a simple constant split rate that is equally applied to all grains.

The intention of this paper is to show one of the pitfalls of numerical simulations that do not include the effect of

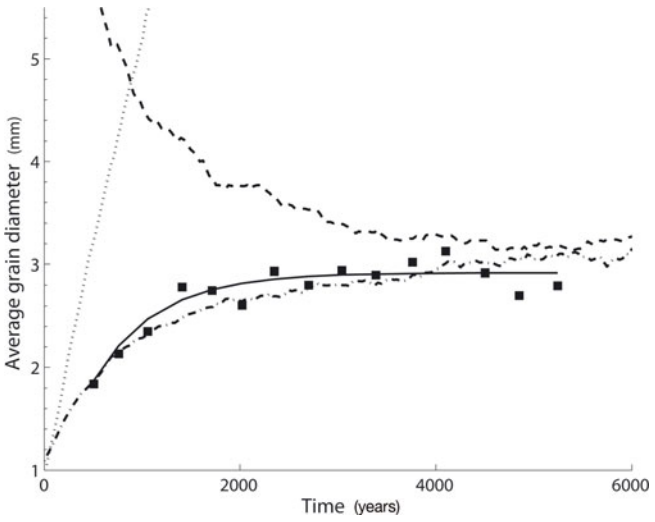


Fig. 5 Evolution of the average grain diameter with time. Static grain growth ($M\gamma = 3.2 \times 10^{-3}$ mm^2 a^{-1}) results in a linear increase of grain diameter (dotted line) (Fig. 4a). Adding a constant split rate ($f = 1.54 \times 10^{-3}$ a^{-1}) for all grains (Fig. 4b) results in the establishment of a stable average grain diameter (dash-dot line). Applying the same settings to an initially large grain microstructure (dashed line) results in the same steady state as for the initially small grain microstructure. For comparison the data from the NorthGRIP core (Fig. 1) have been plotted as well (squares) along with their fit (solid line).

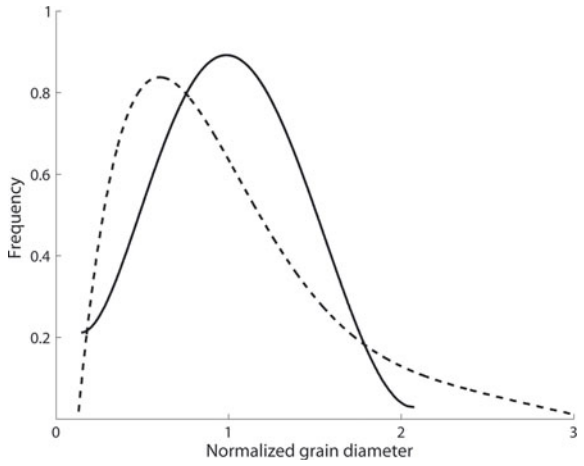


Fig. 6. Normalized frequency distributions of grain diameter. Solid line is the average of 16 simulations of only static grain growth (Fig. 4a). Dashed line is for eight simulations after a steady state has been reached by the competition of static grain growth and splitting (Fig. 4b).

microstructure. The simple analytical model of growth versus splitting produces a curve that can be fitted to data from ice cores. At first sight, it appears that the use of a simple splitting constant, *f*, would be the most problematic simplification. However, our simulations show another simplification that is rarely considered, namely lack of coupling between the growth 'constant', k_0, and *f*. The parameter k_0 is determined by the microstructure. As the microstructure is a variable, k_0 is not a constant, but a variable as well. This observation is of importance because many models that incorporate grain growth, assume k_0 to be constant (Cotterill and Mould, 1976; Randle and others, 1986; Montagnat and Duval, 2000). The numerical simulations show that changing the grain-size distribution from normal to approximately log-normal increases k_0 by a factor of ~3.5. Clearly, other factors may influence k_0, such as grain boundary morphology and grain shape.

The simulation of static grain growth shows that the resulting grain-size distribution is relatively narrow. A normal distribution of measured grain diameters is predicted for static grain growth (Humphreys and Hatherly, 1996). However, grain diameter distributions in ice are usually log-normal, even at relatively shallow depths (Arnaud and others, 1998), for example at 115 m depth in the NorthGRIP core (Thorsteinsson and others, 1997; Svensson and others, 2003). This indicates that the microstructure of ice is already strongly affected by processes other than only static grain growth, well above the transition to a stable grain size. This observation supports the suggestion by various authors (Kipfstuhl and others, 2006, 2009; Durand and others, 2008; Weikusat and others, 2009a,b) that dynamic recrystallization and other processes (Arnaud and others, 2000; Faria and others, 2010) already commence at relatively shallow depth.

The observation that k_0 is dependent on the microstructure may have consequences for the interpretation of grain growth experiments to determine the growth exponent *n*. If the experiment is started with a non-equilibrium microstructure, k_0 may initially be much higher. As the microstructure stabilizes to that characteristic of static grain growth, k_0 decreases (Fig. 7). If the initial phase of

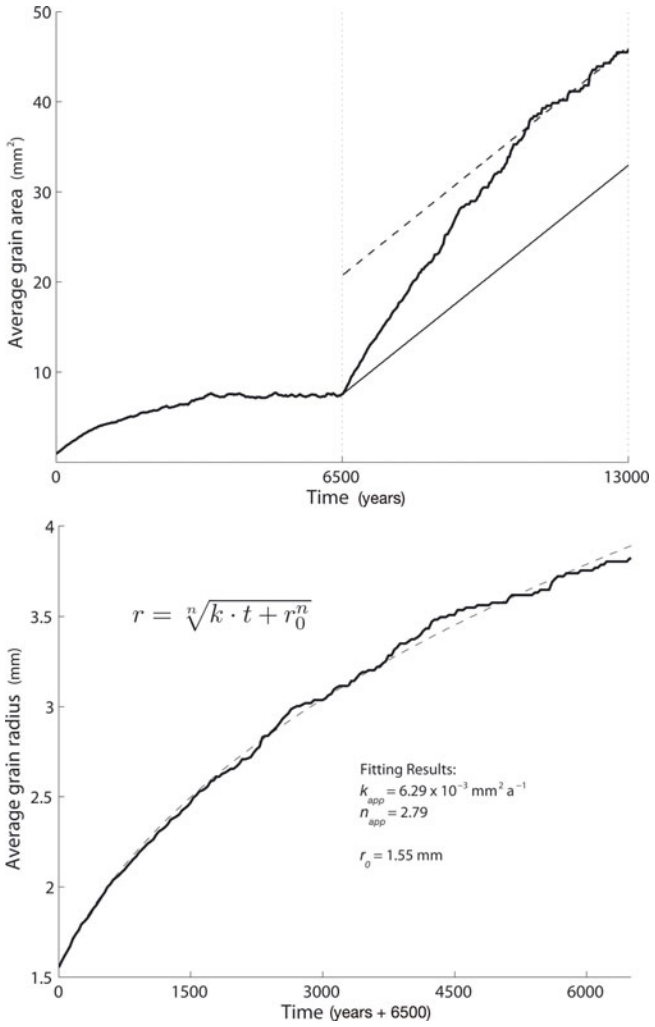


Fig. 7. (a) Grain growth experiment ($M\gamma = 3.2 \times 10^{-3}$ mm^2 a^{-1}) where splitting ($f = 1.54 \times 10^{-3}$ a^{-1}) is turned off after 6500 years. The dashed line shows the growth rate of $k_0 = 1.22$, which is achieved ~4000 years after splitting is stopped, at which stage a foam texture has been established. Just after stopping the splitting, the growth rate is much higher, corresponding to $k_0 = 4.2$. (b) Detailed plot of the experiment in Figure 7a after 6500 years (splitting has been stopped). Equation (1) has been fitted to the experimental curve, giving apparent *k* and *n* values that are incorrect: n_{app} is 2.79 instead of 2 and $k_{app} \pm$ is 6.29×10^{-3} instead of 3.90×10^{-3} mm^2 a^{-1}.

microstructural equilibration is included in an analysis where *k* is assumed to be constant, one would erroneously obtain an exponent *n* that is larger than the real value. For example, the applicable value for k_0 in a polar ice cap would be different from one obtained in a static grain growth experiment, because the microstructure, and hence grain growth in nature, is influenced by additional factors, such as dynamic recrystallization, presence of impurities and bubbles (Cuffey and others, 2000).

CONCLUSIONS

We simulated the process of pure static grain growth and grain growth in competition with another process, namely splitting grains at a constant rate. The numerical simulations show that the growth parameter k_0, normally taken to be a constant, is in fact a function of the microstructure. When

the microstructure is only affected by static grain growth, k_0 is 1.22. The change in microstructure resulting from additional splitting increases k_0 by a factor of ~3.5.

The numerical simulations show that the log-normal grain-size distributions observed in polar ice at shallow depth (≥100 m) are not in accordance with the expected distributions for static grain growth. At least one other process must operate to widen and skew the distribution towards a log-normal distribution. This supports the idea that dynamic recrystallization already operates and influences the microstructure at shallow depth.

The growth exponent and grain boundary properties (surface energy and mobility) are usually determined from experimental growth curves. If the microstructure changes during these experiments, k_0 should not be assumed constant. Making this assumption leads to an overestimate of the growth exponent n.

ACKNOWLEDGEMENT

We gratefully acknowledge funding by the German Research Foundation (DFG) project BO-1776/7.

REFERENCES

Alley, R.B. 1992. Flow-law hypotheses for ice-sheet modeling. *J. Glaciol.,* **38**(129), 245–256.

Alley, R.B., J.H. Perepezko and C.R. Bentley. 1986. Grain growth in polar ice: I. Theory. *J. Glaciol.,* **32**(112), 415–424.

Alley, R.B., A.J. Gow and D.A. Meese. 1995. Mapping *c*-axis fabrics to study physical processes in ice. *J. Glaciol.,* **41**(137), 197–203.

Anderson, M.P. 1986. Simulation of grain growth in two and three dimensions. *In* Hansen, N., D. Juul Jensen, T. Leffers and B. Ralph, *eds. Annealing processes: recovery, recrystallization and grain growth. Proceedings of the 7th Risø International Symposium on Metallurgy and Materials, Roskilde, Denmark*. Roskilde, Risø National Laboratory, 15–34.

Arnaud, L., M. Gay, J.M. Barnola and P. Duval. 1998. Imaging of firn and bubbly ice in coaxial reflected light: a new technique for the characterization of these porous media. *J. Glaciol.,* **44**(147), 326–332.

Arnaud, L., J.M. Barnola and P. Duval. 2000. Physical modeling of the densification of snow/firn and ice in the upper part of polar ice sheets. *In* Hondoh, T., *ed. Physics of ice core records*. Sapporo, Hokkaido University Press, 285–305.

Becker, J.K., P.D. Bons and M.W. Jessell. 2008. A new front-tracking method to model anisotropic grain and phase boundary motion in rocks. *Comput. Geosci.,* **34**(3), 201–212.

Bons, P.D., M.W. Jessell, L. Evans, T. Barr and K. Stüwe. 2001. Modelling of anisotropic grain growth in minerals. *In* Koyi, H.A. and N.S. Mancktelow, *eds. Tectonic modeling: a volume in honor of Hans Ramberg*. Boulder, CO, Geological Society of America, 45–49. (Memoir 193.)

Bons, P.D., D. Koehn and M.W. Jessell, *eds.* 2008. *Microdynamics simulation.* Berlin, Springer-Verlag. (Lecture Notes in Earth Sciences 106.)

Cotterill, P. and P.R. Mould. 1976. *Recrystallization and grain growth in metals.* New York, Wiley.

Cuffey, K.M., T. Thorsteinsson and E.D. Waddington. 2000. A renewed argument for crystal size control of ice sheet strain rates. *J. Geophys. Res.,* **105**(B12), 27,889–27,894.

De La Chapelle, S., O. Castelnau, V. Lipenkov and P. Duval. 1998. Dynamic recrystallization and texture development in ice as revealed by the study of deep ice cores in Antarctica and Greenland. *J. Geophys. Res.,* **103**(B3), 5091–5105.

Durand, G. *and 10 others*. 2006. Effect of impurities on grain growth in cold ice sheets. *J. Geophys. Res.,* **111**(F1), F01015. (10.1029/2005JF000320.)

Durand, G., A. Perrson, D. Samyn and A. Svensson. 2008. Relation between neighbouring grains in the upper part of the NorthGRIP ice core – implications for rotation recrystallization. *Earth Planet. Sci. Lett.,* **265**(3–4), 666–671.

Duval, P. 2000. Deformation and dynamic recrystallization of ice in polar ice sheets. *In* Hondoh, T., *ed. Physics of ice core records*. Sapporo, Hokkaido University Press, 103–113.

Duval, P. and O. Castelnau. 1995. Dynamic recrystallization of ice in polar ice sheets. *J. Phys. IV [Paris],* **5**(C3), 197–205.

Duval, P., M.F. Ashby and I. Anderman. 1983. Rate-controlling processes in the creep of polycrystalline ice. *J. Phys. Chem.,* **87**(21), 4066–4074.

Faria, S.H. and S. Kipfstuhl. 2004. Preferred slip-band orientations and bending observed in the Dome Concordia (East Antarctica) ice core. *Ann. Glaciol.,* **39**, 386–390.

Faria, S.H., D. Ktitarev and K. Hutter. 2002. Modelling evolution of anisotropy in fabric and texture of polar ice. *Ann. Glaciol.,* **35**, 545–551.

Faria, S.H., J. Freitag and S. Kipfstuhl. 2010. Polar ice structure and the integrity of ice-core paleoclimate records. *Quat. Sci. Rev.,* **29**(1–2), 338–351.

Glazier, J.A., S.P. Gross and J. Stavans. 1987. Dynamics of two-dimensional soap froths. *Phys. Rev. A,* **36**(1), 306–312.

Gow, A.J. 1969. On the rates of growth of grains and crystals in South Polar firn. *J. Glaciol.,* **8**(53), 241–252.

Gow, A.J. and T. Williamson. 1976. Rheological implications of the internal structure and crystal fabrics of the West Antarctic ice sheet as revealed by deep core drilling at Byrd Station. *Geol. Soc. Am. Bull.,* **87**(12), 1665–1677.

Gow, A.J. *and 6 others*. 1997. Physical and structural properties of the Greenland Ice Sheet Project 2 ice cores: a review. *J. Geophys. Res.,* **102**(C12), 26,559–26,575.

Herron, S.L. and C.C. Langway, Jr. 1982. A comparison of ice fabrics and textures at Camp Century, Greenland and Byrd Station, Antarctica. *Ann. Glaciol.,* **3**, 118–124.

Houseman, G., T. Barr and L. Evans. 2008. Basil: stress and deformation in a viscous material. *In* Bons, P.D., D. Koehn and M.W. Jessell, *eds. Microdynamics simulation*. Berlin, Springer-Verlag, 139–154. (Lecture Notes in Earth Sciences 106.)

Humphreys, F.J. and M. Hatherly. 1996. *Recrystallization and related annealing phenomena.* Oxford, Pergamon.

Jessell, M.W. and P.D. Bons. 2002. The numerical simulation of microstructure. *In* de Meer, S., M.R. Drury, J.H.P. de Bresser and G.M. Pennock, *eds. Deformation mechanisms, rheology and tectonics: current status and future perspectives*. London, Geological Society, 137–147. (Special Publication 200.)

Jessell, M., P. Bons, L. Evans, T. Barr and K. Stüwe. 2001. Elle: the numerical simulation of metamorphic and deformation microstructures. *Comput. Geosci.,* **27**(1), 17–30.

Jessell, M.W., E. Siebert, P.D. Bons, L. Evans and S. Piazolo. 2005. A new type of numerical experiment on the spatial and temporal patterns of localization of deformation in a material with a coupling of grain size and rheology. *Earth Planet. Sci. Lett.,* **239**(3–4), 309–326.

Kipfstuhl, S. *and 6 others*. 2006. Microstructure mapping: a new method for imaging deformation-induced microstructural features of ice on the grain scale. *J. Glaciol.,* **52**(178), 398–406.

Kipfstuhl, S. *and 8 others*. 2009. Evidence of dynamic recrystallization in polar firn. *J. Geophys. Res.,* **114**(B5), B05204. (10.1029/2008JB005583.)

Lipenkov, V.Ya., N.I. Barkov, P. Duval and P. Pimienta. 1989. Crystalline texture of the 2083 m ice core at Vostok Station, Antarctica. *J. Glaciol.,* **35**(121), 392–398.

Mathiesen, J. *and 6 others*. 2004. Dynamics of crystal formation in the Greenland NorthGRIP ice core. *J. Glaciol.,* **50**(170), 325–328.

Montagnat, M. and P. Duval. 2000. Rate controlling processes in the creep of polar ice: influence of grain boundary migration associated with recrystallization. *Earth Planet. Sci. Lett.,* **183**(1–2), 179–186.

Morland, L.W. 2009. Age–depth correlation, grain growth and dislocation-density evolution, for three ice cores. *J. Glaciol.,* **55**(190), 345–352.

Mullins, W.W. 1989. Estimation of the geometrical rate constant in idealized three dimensional grain growth. *Acta Metall.,* **37**(11), 2979–2984.

Piazolo, S., P.D. Bons, P.D. Jessell, L. Evans and C.W. Passchier. 2002. Dominance of microstructural processes and their effect on microstructural development: insights from numerical modelling of dynamic recrystallization. *In* de Meer, S., M.R. Drury, J.H.P. de Bresser and G.M. Pennock, *eds. Deformation mechanisms, rheology and tectonics: current status and future perspectives.* London, Geological Society. (Special Publication 200.)

Piazolo, S., M.W. Jessell, D.J. Prior and P.D. Bons. 2004. The integration of experimental in-situ EBSD observations and numerical simulations: a novel technique of microstructural process analysis. *J. Microsc.,* **213**(3), 273–284.

Placidi, L., S.H. Faria and K. Hutter. 2004. On the role of grain growth, recrystallization and polygonization in a continuum theory for anisotropic ice sheets. *Ann. Glaciol.,* **39**, 49–52.

Randle, V., B. Ralph and N. Hansen. 1986. Grain growth in crystalline materials. *In* Hansen, N., D. Juul Jensen, T. Leffers and B. Ralph, *eds. Annealing processes: recovery, recrystallization and grain growth. Proceedings of the 7th Risø International Symposium on Metallurgy and Materials, Roskilde, Denmark.* Roskilde, Risø National Laboratory, 123–142.

Read, W.T. 1953. *Dislocations in crystals.* New York, McGraw-Hill.

Smith, C.S. 1964. Some elementary principles of polycrystalline microstructure. *Metall. Rev.,* **9**(33), 1–48.

Svensson, A. *and 6 others.* 2003. Properties of ice crystals in NorthGRIP late- to middle-Holocene ice. *Ann. Glaciol.,* **37**, 113–122.

Thorsteinsson, T., J. Kipfstuhl and H. Miller. 1997. Textures and fabrics in the GRIP ice core. *J. Geophys. Res.,* **102**(C12), 26,583–26,599.

Urai, J.L., W.D. Means and G.S. Lister. 1986. Dynamic recrystallization of minerals. *In* Hobbs, B.E. and H.C. Heard, *eds. Mineral and rock deformation: laboratory studies: the Paterson Volume.* Washington, DC, American Geophysical Union, 161–199. (Geophysical Monograph 36.)

Weaire, D. and N. Rivier. 2009. Soap, cells and statistics – random patterns in two dimensions. *Contemp. Phys.,* **50**(1), 199–239.

Weikusat, I., S. Kipfstuhl, N. Azuma, S.H. Faria and A. Miyamoto. 2009a. Deformation microstructures in an Antarctic ice core (EDML) and in experimentally deformed artificial ice. *In* Hondoh, T., *ed. Physics of ice core records II.* Sapporo, Hokkaido University Press, 115–123. (Low Temperature Science Supplement Issue 68.)

Weikusat, I., S. Kipfstuhl, S.H. Faria, N. Azuma and A. Miyamoto. 2009b. Subgrain boundaries and related microstructural features in EDML (Antarctica) deep ice core. *J. Glaciol.,* **55**(191), 461–472.

Weikusat, I., A. Miyamoto, S.H. Faria, S. Kipfstuhl, N. Azuma and T. Hondoh. 2011. Subgrain boundaries in Antarctic ice quantified by X-ray Laue diffraction. *J. Glaciol.,* **57**(201), 111–120.

Weygand, D., Y. Bréchet, J. Lépinoux and W. Gust. 1998. Three dimensional grain growth: a vertex dynamics simulation. *Philos. Mag. B,* **79**(5), 703–716.

APPENDIX: DERIVATION OF EQUATIONS (4) AND (6)

The number, N, of grains per unit volume equals

$$N = \frac{1}{ar^3}, \tag{A1}$$

where a is a shape factor that depends on the shape of grains. If only static grain growth operates, Equation (A1) can be combined with Equation (1), which gives, assuming $n = 2$:

$$N = \frac{1}{a\left(kt + r_0^2\right)^{3/2}} \iff \frac{\mathrm{d}N}{\mathrm{d}t} = \frac{-3ka^{2/3}}{2} N^{5/3}. \tag{A2}$$

Adding the effect of splitting Equation (3) has an additional term, and the number of grains per time is

$$\frac{\mathrm{d}N}{\mathrm{d}t} = -\alpha N^{5/3} + fN, \text{ and hence } -\int \frac{\mathrm{d}N}{\alpha N^{5/3} - fN} = \int \mathrm{d}t, \tag{A3}$$

where $\alpha = 3ka^{2/3}/2$.

This equation can be solved with the standard indefinite integral:

$$\int \frac{\mathrm{d}x}{x(x^p - b^p)} = \frac{1}{pb^p} \ln\left(\frac{x^p - b^p}{x^p}\right). \tag{A4}$$

By using $p = 2/3$, $f = N$ and $b = (f/\alpha)^{3/2}$ the relation between t and N results in

$$t = \frac{-3}{2f} \ln\left(\frac{N^{2/3} - \frac{f}{\alpha}}{N^{2/3}}\right) \iff N^{-2/3} = \frac{\alpha}{f}\left(1 - \mathrm{e}^{\frac{-2f}{3}t}\right), \tag{A5}$$

and by using Equation (A1) grain-size evolution finally gives

$$r^2 = \frac{3k}{2f}\left(1 - \mathrm{e}^{\frac{-2f}{3}t}\right). \tag{A6}$$

Note that variable α is replaced by full expression (A3), and the shape factor, a, used in Equation (A1) is cancelled out of the equation.

The derivation of Equation (6) for two dimensions is similar to the above. In two dimensions, Equation (1) still holds and if $n = 2$ we can write for the mean grain area, A:

$$A - A_0 = kt. \tag{A7}$$

The number, N, of grains per unit area equals $1/A$, which gives

$$N = \frac{1}{kt + A_0}. \tag{A8}$$

Taking the time derivative and adding the increase in number of grains as a result of constant splitting, Equation (3) results in:

$$\frac{\mathrm{d}N}{\mathrm{d}t} = \frac{-k}{(kt + A_0)^2} + fN = -kN^2 + fN. \tag{A9}$$

The last equation can be solved with the indefinite integral of Equation (A4) to obtain:

$$\frac{1}{N} = A = \frac{k}{f}\left(1 - \mathrm{e}^{-ft}\right). \tag{A10}$$

MS received 31 January 2011 and accepted in revised form 25 July 2011

APPENDIX 2

INFLUENCE OF BUBBLES ON GRAIN GROWTH IN ICE

Journal of Structural Geology xxx (2012) 1–10

Contents lists available at SciVerse ScienceDirect

Journal of Structural Geology

journal homepage: www.elsevier.com/locate/jsg

Influence of bubbles on grain growth in ice

Jens Roessiger [a], Paul D. Bons [a,*], Sérgio H. Faria [b,c]

[a] Department for Geosciences, Eberhard Karls University, Wilhelmstraße 56, 72074 Tübingen, Germany
[b] Basque Centre for Climate Change (BC3), Alameda Urquijo 4, 48008 Bilbao, Spain
[c] Ikerbasque, Basque Foundation for Science, 48011 Bilbao, Spain

ARTICLE INFO

Article history:
Received 29 March 2012
Received in revised form
16 October 2012
Accepted 11 November 2012
Available online xxx

Keywords:
Grain growth
Growth exponent
Growth constant
Ice
Numerical simulations
Two phase

ABSTRACT

Numerical static grain growth simulations of ice with air bubbles as a second phase show a significant drop in grain-growth rate compared to bubble-free ice. The magnitude of this drop in growth rate is dependent on the bubble boundary mobility, the volume fraction of air, the average bubble size and the bubble size distribution. The rate of grain growth decreases at first, as the microstructure evolves towards a steady state. Only then does grain growth follow the expected linear increase of mean grain area with time. In experiments, this decrease in growth rate could erroneously be interpreted as growth with a deviating growth exponent.

1. Introduction

Natural ice is rarely a single-phase material. It generally contains chemical impurities and dust, as well as air inclusions. In glaciers and ice sheets, air is trapped in the form of bubbles during the compaction from snow to firn to ice (Arnaud et al., 2000). In lake- or sea-ice, air bubbles may also occur due to gas accumulations along the water–ice interface (Schulson and Duval, 2009). With ongoing burial, air bubbles are compressed and may eventually convert to clathrates, in a transition zone that for polar ice sheets lies approximately in the range 600–1200 m depth (Barnes et al., 2002; Faria et al., 2009; Hondoh, 2009; Lipenkov et al., 1992). Air bubbles, clathrates, dust and other chemical impurities all influence recrystallisation of ice (Durand et al., 2006; Faria et al., 2010). While one expects that small particles such as microscopic inclusions and clathrates may mostly modify the grain boundary velocity, air bubbles form a significant volume fraction in the upper few hundred meters of ice sheets and glaciers, and could therefore influence recrystallisation even more (Arena et al., 1997; Azuma et al., 2012).

Grain size increases significantly in the upper few hundred metres of polar ice sheets (De La Chapelle et al., 1998). The increase in grain size is assumed to be driven by a reduction of grain boundary surface energy, a process usually termed static or normal grain growth (Alley et al., 1986a; Smith, 1964). This process is thought to dominate over flow-induced dynamic recrystallisation or polygonisation (Urai et al., 1986), which increasingly affects the ice microstructure with depth (Alley, 1992; Duval and Castelnau, 1995; Faria et al., 2002). The depth at which dynamic recrystallisation becomes significant is still under debate (Kipfstuhl et al., 2009). The stabilisation of grain size at depth has been observed in several deep ice cores. It is thought to result from a balance between grain growth and grain size reduction by dynamic recrystallisation (Gow et al., 1997; Gow and Williamson, 1976; Mathiesen et al., 2004; Montagnat and Duval, 2000; Thorsteinsson et al., 1997). Such a dynamic equilibrium between grain size increase and decrease is also invoked to explain grain sizes in other minerals, for example olivine, in deforming rocks (Herwegh and Handy, 1996; De Bresser et al., 2001).

Knowledge of the rate of grain size increase by grain growth is of clear relevance to be able to interpret grain sizes, grain size evolution and microstructures (Urai et al., 1986; Stöckhert and Duyster, 1999; Herwegh and Berger, 2003; Herwegh et al., 2011; etc.). The increase of grain size (diameter D_t) with time (t) from an initial size (D_0) is usually expressed in the form (Anderson, 1986; Glazier et al., 1987; Evans et al., 2001):

* Corresponding author. Tel.: +49 70712976469; fax: +49 7071293060.
E-mail addresses: roessiger@gmail.com (J. Roessiger), paul.bons@uni-tuebingen.de (P.D. Bons), sergio.faria@bc3research.org (S.H. Faria).

0191-8141/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.jsg.2012.11.003

$$D_t^n - D_0^n = kt \text{ with } D_t^n \approx kt (\text{if } D_0 << D_t) \quad (1)$$

The growth parameter (k) is determined by the grain boundary energy (γ) and the grain boundary mobility (M) with $k = k_0 M\gamma$. M [m^2s/kg] and γ [kg/s^2] are material properties. The temperature dependence of k is determined by the activation energy Q (since $k \propto \exp^{-Q/RT}$, with R the universal gas constant and T the absolute temperature). In the literature, Q is assumed to be about 40–50 kJ/mol, which is based on observations on polar ice sheets (Gow, 1969, 1971; Paterson, 1994). Recent experiments by Azuma et al. (2012) indicate that Q for pure ice is much higher at about 110–120 kJ/mol.

The dimensionless parameter k_0 is usually assumed constant with a theoretical value of $k_0 = 4.48$ or 2 in a two- or three-dimensional aggregate, respectively (Mullins, 1989). However, in reality k_0 is not a constant, but depends on the microstructure (Arena et al., 1997; Roessiger et al., 2011). Only if the microstructure is a foam texture will k_0 equal the theoretical value. The growth exponent (n) depends on the grain growth mechanism (Evans et al., 2001). In a pure grain aggregate, grain growth is controlled by the curvature of grain boundaries. If all boundaries have the same mobility, the growth exponent should be two. This can be derived from a simple dimension analysis, considering that the unit of k is m^2/s. Inserting this in Eq. (1) gives $n = 2$.

In impure grain aggregates, grain growth is a much more complicated process, due to the interaction between the grains and the impurities (Herwegh et al., 2011 and references therein). Impurities, such as dust particles, chemical impurities, or second phases such as air bubbles, may hinder or completely stop grain boundary movement (Zener pinning; Olgaard and Evans, 1986, 1988; Brodhag and Herwegh, 2010). If impurities inhibit grain boundary movement, growth comes to a complete halt when all boundaries are pinned (Weygand et al., 1999; Herwegh et al., 2011). In this case equation (1) does not apply, but grain size will asymptotically approach a fully pinned state. The maximum grain size (D_{max}) is usually related to the fraction of second phase (f) and the size of the second phase particles or regions (d_s) by the Zener equation, where z is a scaling parameter:

$$D_{max} = z\frac{d_s}{f^m} \quad (2)$$

See Olgaard and Evans (1986), Manohar et al. (1998) and Evans et al. (2001) for the background of the Zener equation and variations proposed in the literature.

If particles can be dragged along by the boundaries (Zener drag), the boundaries keep moving, but at a reduced rate as they accumulate more and more particles. If the second phase occupies a significant fraction of the material, as is the case for air bubbles in ice, the overall growth rate is assumed to be controlled by the increase in d_s of the minor phase (Hiraga et al., 2010a). In a two-phase material, such as ice with air bubbles, grains and bubbles represent phase regions. Isolated phase regions (air bubbles) can grow by two basic mechanisms:

- Diffusional material transfer between phase regions. The driving force for this is the higher surface energy of small compared to large phase regions, which have a larger radius of curvature (Ostwald ripening). If transport is by volume diffusion, theory predicts a growth exponent $n = 3$, while $n = 4$ is expected for grain-boundary diffusion (Evans et al., 2001). However, much higher growth exponents have been reported in the literature (Hiraga et al., 2010a; Ohuchi and Nakamura, 2007; Olgaard and Evans, 1988; Tullis and Yund, 1982; Yamazaki et al., 1996).
- Migration and merging of phase regions. Grain boundaries may drag phase regions, which may lead to them merging to form larger volumes (Brodhag and Herwegh, 2010). No diffusional exchange between phase regions is required for this mechanism.

Depending on the mechanism of migration of the second phase, surface-energy driven grain growth does not necessarily follow the normal grain growth law (Eq. (1)). For example, in case of migration of bubbles in ice, the migration rate is a function of bubble radius, diffusivity of water molecules in air, etc. (Hsueh et al., 1982; Alley et al., 1986a). In this case, the growth rate (dD/dt) is no longer proportional to $1/D$ (Eq. (12) in Azuma et al., 2012) and if Eq. (1) were to be applied, high apparent growth exponents are the result.

Few studies on grain growth in ice specifically address the influence of air bubbles (Arena et al., 1997; Azuma et al., 2012). It is usually assumed that grain boundary migration in nature is in the fast migration regime, also called regime 2 (Alley et al., 1986b). In this regime, migrating boundaries can sweep across bubbles and these do not remain on the boundaries but slow them down. The inferred regime 2 migration is based on the observation that air bubbles occur inside ice grains (Alley et al., 1986b). However, in the upper part of polar ice sheets, around the firn–ice transition, most bubbles are actually residing on grain boundaries (Arnaud et al., 1998; Kipfstuhl et al., 2009), which suggests that boundaries can usually not sweep across bubbles and leave them behind.

In this paper we investigate grain growth in ice with air bubbles with numerical simulations. Our model only includes grain boundary migration driven by the reduction in grain boundary curvature and thus excludes air transfer between air bubbles (Ostwald ripening). Rather than attempting to provide a grain-growth law for ice, as a function of parameters such as bubble content, temperature, etc., we discuss the behaviour of a two-phase grain aggregate, with particular attention to the influence of the two-phase (ice–air) boundary mobility relative to the single-phase (ice–ice) boundary mobility.

2. Method

For our simulations we used the open-source modelling software package Elle (Bons et al., 2008; Jessell et al., 2001; Piazolo et al., 2010). It has been used for the simulation of recrystallisation processes in ice, rock-forming minerals, and partially molten rocks (Becker et al., 2008; Bons et al., 2001; Jessell et al., 2003; Piazolo et al., 2004; Roessiger et al., 2011). The 2-dimensional microstructure is defined by a contiguous set of polygons (termed *flynns*) that are themselves defined by boundary nodes (termed *bnodes*). Bnodes are linked to two or three neighbours by straight segments (Fig. 1). Spatial resolution is defined by the *switch distance* (D_{sw}), here set at 0.005 of the unit-sized square model. Spacing between bnodes is held between 1 and 2.2 × D_{sw} by either inserting or removing bnodes when they are too far apart or too close, respectively. With a starting grain aggregate of about 420 grains, this means that grains have on average about 26 bnodes. A neighbour switch is induced when two converging grain boundary triple junctions are less than D_{sw} apart. Attributes can be assigned to both boundaries and flynns. Flynns represent individual ice grains or air bubbles. Boundaries can be (1) ice–ice grain boundaries, (2) ice–air interfaces, or (3) air–air boundaries, which are sometimes necessary for numerical reasons, but have no physical meaning. Periodic boundary conditions are applied in both horizontal and vertical directions, meaning that a grain boundary that reaches the left or bottom edge continues its motion on the right or top edge, respectively. The model can thus be considered a unit cell in an infinite grain aggregate.

Please cite this article in press as: Roessiger, J., et al., Influence of bubbles on grain growth in ice, Journal of Structural Geology (2012), http://dx.doi.org/10.1016/j.jsg.2012.11.003

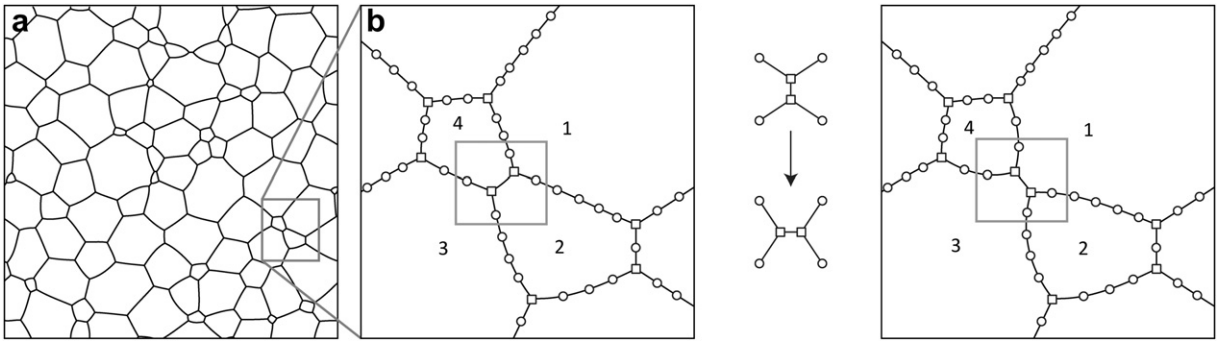


Fig. 1. (a) The Elle data structure consists of polygons, termed flynns. (b) In detail the boundaries consist of nodes (bnodes) which can have either two or three neighbours (circles and squares, respectively). If two triple nodes get closer together than the switch distance (e.g. when polygons 2 and 4 are shrinking), a neighbour switch is induced. After the switch polygons 2 and 4 are no longer neighbours and can shrink further. Double nodes are removed or inserted if they are too close or too far apart.

2.1. Movement of single-phase ice–ice boundaries

The model aims to simulate grain growth that is driven by a reduction of surface energy, based on the algorithm of Becker et al. (2008). No air transfer by diffusion between individual bubbles is incorporated in the model. Each program cycle, each bnode is moved over a small distance, representing the boundary movement during a small time increment. For ice–ice boundaries, movement is solely determined by the reduction of surface energy. For a given bnode, four orthogonal trial positions at very small distances from the original bnode position are carried out and the total boundary length for these positions is calculated. The total free energy ($E_{(j)}$) for each j-th trial position is simply the product of the distances (S) to the two or three neighbouring bnodes, multiplied by the ice–ice surface energy (γ_{ii}), here set at 0.065 J/m^2 (Ketcham and Hobbs, 1969):

$$E_{(j)} = \gamma_{ii} \sum S_{(j)} \tag{3}$$

The driving force for boundary migration is the spatial gradient in free energy, which can be calculated from the four $E_{(j)}$ values. The direction of bnode movement is thus determined by calculating the direction of highest negative free-energy gradient (Fig. 2a). The velocity (v) of a boundary is proportional to the driving stress (σ) and the boundary mobility (M_{ii}), another material property that is modified for several experiments. The driving stress is calculated from the driving force, by taking into account the length and orientations of the boundary segments this force acts on. The third dimension is assumed to be unity which then drops out from the equation again. The bnode is finally moved over a small distance (Δx) for a small time increment (Δt) with:

$$\Delta x = M_{ii} \cdot \sigma \cdot \Delta t \tag{4}$$

A more detailed description of this node movement algorithm can be found in Becker et al. (2008) and Bons et al. (2008).

2.2. Movement of two-phase ice–air boundaries

For the two-phase (ice–air) boundaries an additional factor that influences boundary velocity and direction is the conservation of mass requirement. In the model, with no mass transfer between air bubbles, this implies conserving the cross-sectional area of each bubble. Movement of a single ice–air bnode, however, normally involves changing the areas of the adjacent air and ice flynns and, therefore, a small violation of the conservation of mass requirement. Not allowing this would freeze the ice–air bnodes in the model. In reality, cross-sectional area changes by inward bubble surface migration at one side of a bubble would be compensated by outward migration at another side. Since bnode movements are

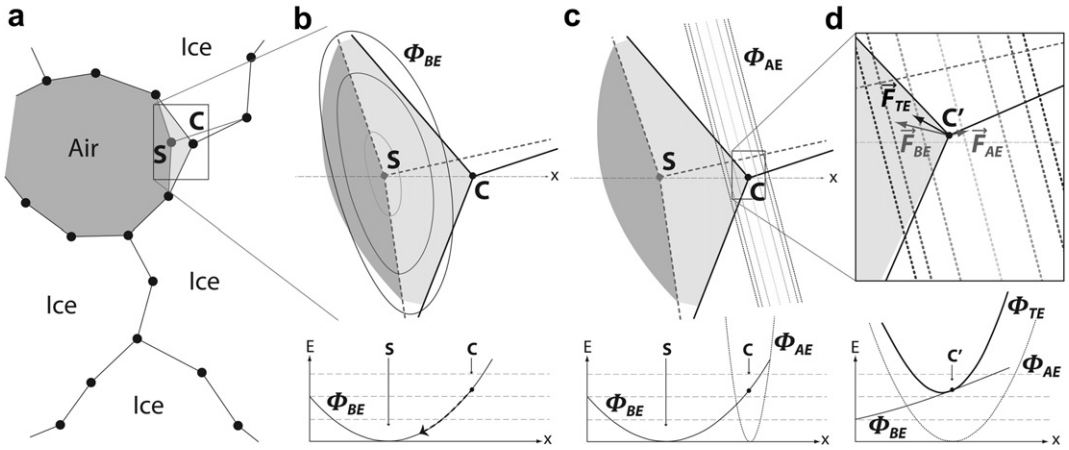


Fig. 2. Explanation of the node movement routine. (a) Local geometry around node (C) that will be moved. Nodes are moved one by one, while all other nodes remain fixed during a single node movement. (b) The boundary-energy (Φ_{BE}) field around node C. With the current surface energy settings, the lowest energy position for the node is at location S. The lower part shows the free energy (E) profile along the x-axis (dashed line). The free-energy gradient causes movement of the node in the direction of S. (c) The area-energy (Φ_{AE}) field around a node at C. Only movement of the node along a line parallel to the line connecting the neighbour ice–air boundary nodes is area conservative. Different movement will change the area and hence the free energy of the system, as illustrated in the free-energy profile. (d) The combination of both energy fields gives the total energy (Φ_{TE}) field, which drives the node down the steepest gradient of this field (arrow $\vec{F}_{TE}$).

calculated and carried out one by one, an additional energy term needs to be incorporated in Eq. (3).

Again, we take four trial positions and calculate their effect on total surface energy (E_{surf}). However (γ_{ii}) is now different for ice–ice and ice–air boundaries. Additionally, the bubble cross-sectional area ($A_{(j)}$) is calculated every cycle. It results from bnode movement, and is normalised by the original area (A_0) of the bubble, which is stored as an attribute of the air flynn. The total free energy for each of j trial positions is now calculated with:

$$E_{(j)} = E_{\mathrm{surf}(j)} + a\left(\frac{A_{(j)} - A_0}{A_0}\right)^b \quad (5)$$

The parameter a (set to 0.01) determines the magnitude of the contribution of bubble area changes to the total surface energy. The parameter b (set to 2) determines the sensitivity of the energy to changes in the bubble original cross-sectional area. This routine effectively simulates the compressibility of air in a bubble. Surface energy drives inward movement of the bubble surface, which would compress the enclosed air and increase the pressure. The pressure increasingly counteracts shrinkage of the bubble, until equilibrium is reached and the bubble overpressure balances the surface tension (Fig. 2b). Once the free energy gradient of the ice–air bnode is calculated with Eq. (5), the bnode movement is calculated in the same way as for an ice–ice bnode, but taking into account the different mobilities of the ice–ice and ice–air boundaries.

2.3. Topological events

Starting with an initial microstructure (Fig. 3), the program goes through 80,000 cycles. At the beginning of each cycle, the index number of each bnode is randomized. Next, the program goes through this randomized list of bnodes and treats each one by one. This ensures that each bnode is treated once, but in a different order every time step. For each bnode, displacement is calculated and carried out immediately. After each movement, the program carries out a number of checks. It first determines whether the displacement is permissible, i.e. does not cause boundary segments to cross each other. The second check is whether topological changes need to be carried out. If two triple junctions approach each other to less than the switch distance (D_{sw}), a neighbour switch (Fig. 1b) is induced by re-ordering the links to neighbouring bnodes. A grain is removed if it consists of only three triple junctions and is below a set threshold in size. With individual grains disappearing during a simulation, the average grain size increases with time.

Neighbour switches can lead to two air flynns becoming neighbours, which leads to the creation of an air–air boundary, without physical meaning. The air bubble now consists of a cluster of flynns. The original cross-sectional area (A_0) of the new, merged bubble is the sum of the original areas of the two merged flynns. This value is attributed to all flynns in the cluster. In the simulations, air–air boundaries are treated in the same way as ice–ice boundaries. They are given a very low surface energy (20 times lower than γ_{ii}), so that they do not influence the shape of air bubbles. A high mobility (1.2 times the ice-air boundary mobility) ensures that air–air bnodes/segments/boundaries can still move (and usually disappear in the end) and do not control the mobility of bubble-surface bnodes, as the boundary velocity is controlled by the least mobile segments.

2.4. Settings

Grain boundary surface energy of ice–ice boundaries is reasonably well constrained at about 0.065 J/m^2 (Ketcham and Hobbs, 1969). Considering that the ice lattice is highly anisotropic, it is to be expected that the surface energy is actually a function of its orientation relative to the two adjacent crystal lattice orientations (Bons et al., 2001). The equigranular shape of statically recrystallised ice grains, however, suggests that surface energy anisotropy is not strong in ice. For simplicity, this effect was therefore ignored and all ice–ice boundaries were given the constant surface energy of 0.065 J/m^2. The surface energy of ice–air boundaries cannot be constrained well from the literature. However, the spherical shape of air bubbles in ice indicates a high dihedral angle at ice–air grain boundary triple junctions. The dihedral angle (ω) is a function of the ice–ice and ice–air surface energies (γ_{ii} and γ_{ia}):

$$\omega = 2\cdot\cos^{-1}\left(\frac{\gamma_{ii}}{2\cdot\gamma_{ia}}\right) \quad (6)$$

In all simulations, γ_{ia} was set at eight times (0.52 J/m^2) that of ice–ice boundaries, giving a dihedral angle of 173° and almost circular air bubbles.

Azuma et al. (2012) experimentally determined the growth constant for pure ice down to −40 °C. Using their 113 kJ/mol activation energy, k at −32 °C is 7.9 10^{-14} m^2/s, which is one or two orders of magnitude higher than values commonly used in the literature (Petit et al., 1987; Paterson, 1994; Thorsteinsson et al., 1997; Arena et al., 1997). The temperature of −32 °C was chosen as it was the temperature used by Mathiesen et al. (2004) for their modelling of grain size evolution in the upper 880 m of the North Greenland Icecore Project (NorthGRIP). Using $k_0 = 2$ (for three-dimensional growth) and $\gamma_{ii} = 0.065$ J/m^2, this gives $M_{ii} = 6.07\cdot10^{-13}$ m^2 s/kg. We set the model size at 4 × 4 cm, giving a starting grain diameter of about 2 mm, again comparable to the NorthGRIP data. To achieve the same growth rate, k, at the same γ_{ii} in our 2-dimensional simulations ($k_0 = 4.48$), we use a time step of 0.79 h in the calculations and a numerical mobility of ice–ice boundaries of $M_{ii} = 2.70\cdot10^{-13}$ m^2 s/kg. For the mobility of ice–air boundaries (M_{ia}) we ran a series of simulations at different ratios $R = M_{ia}/M_{ii}$ of 10, 1, 0.1 and 0.01. This way, we could investigate the effect of very immobile versus very mobile ice–air boundaries, relative to ice–ice boundaries.

Three starting models were used for the simulations. The first is a foam texture with 420 grains and no air bubbles. This model serves as a reference to determine the grain growth parameter (k in Eq. (1)). The second model is the same foam texture with 530 equal-sized bubbles occupying an area fraction of 7%, while the third model has 134 equal-sized bubbles and an area fraction of 14%. The main difference between models 2 and 3 is that many ice–ice grain boundaries in model 3 are bubble-free, while virtually all ice–ice boundaries in model 2 carry one or more bubbles. It should be noted that the model does not allow flynns within flynns, and hence all bubbles are and remain on ice–ice boundaries. The numerical model can clearly be scaled to any size (with appropriate concomitant scaling of the time step). Here we set the model size at 40 × 40 mm, which means that the average initial ice grain diameter is 2 mm and that of the bubbles is 0.5 mm and 1.4 mm in models 2 and 3, respectively.

3. Results

Simulations for pure, bubble-free ice show a steady increase in grain size, with D^2 proportional to time in accordance with Eq. (1) (Fig. 3a and Fig. 4 growth). The microstructure started as and remained a foam texture with approximately 120° angles at triple junctions and smoothly curved grain boundaries. This reference experiment shows that the energy minimisation routine indeed

Please cite this article in press as: Roessiger, J., et al., Influence of bubbles on grain growth in ice, Journal of Structural Geology (2012), http://dx.doi.org/10.1016/j.jsg.2012.11.003

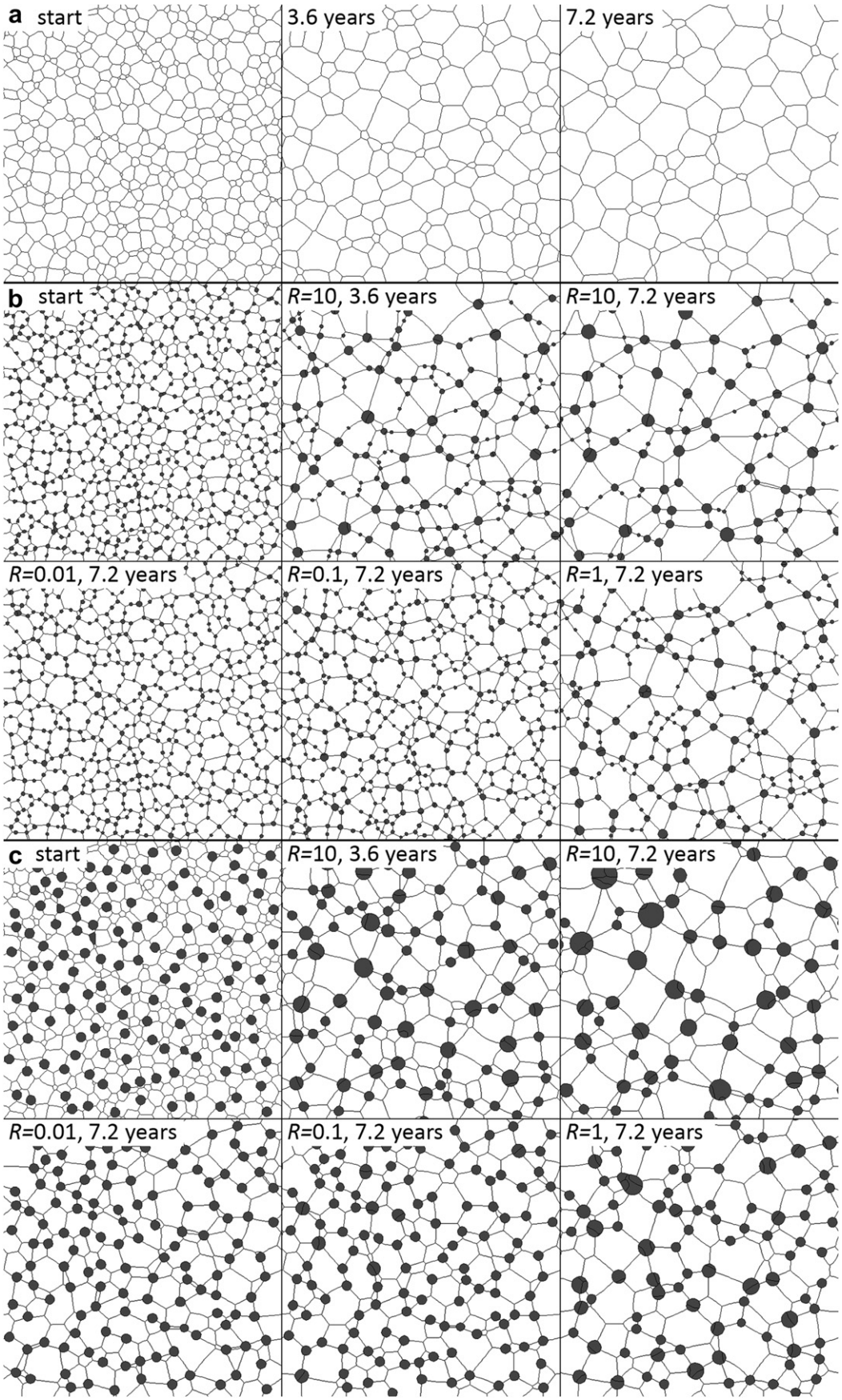


Fig. 3. (a) Three stages of a simulation of growth in bubble-free ice. (b) Results of the numerical simulation with model 2, where bubbles are much smaller than the average ice grain size. Top row shows the starting microstructure and two stages of growth for a high mobility ratio (R) of 10. Bottom rows show the final stage for different mobility ratios. Drag and merging of bubbles at high mobility ratio reduces the number of bubbles and increases their size and size variation. (c) Same as (b), but for model 3 with fewer, but larger bubbles.

Please cite this article in press as: Roessiger, J., et al., Influence of bubbles on grain growth in ice, Journal of Structural Geology (2012), http://dx.doi.org/10.1016/j.jsg.2012.11.003

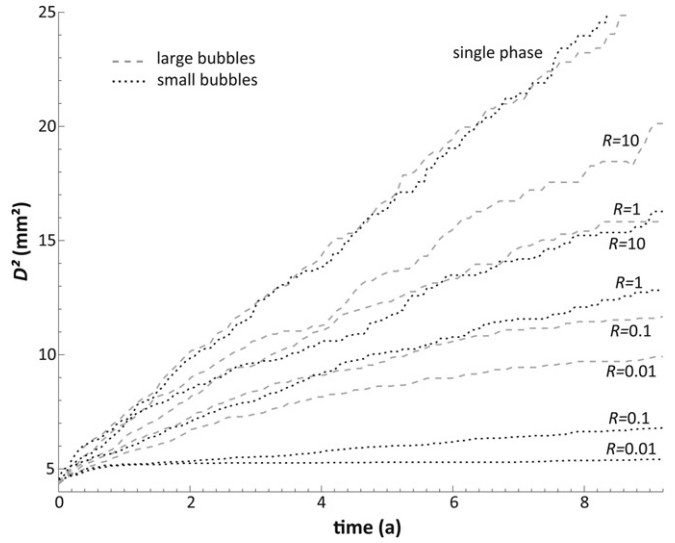


Fig. 4. Mean grain size evolution of all experiments, shown as squared grain diameter (D^2) against time in years. The single-phase simulations show the expected linear increase in grain area. The second phase slows down the growth rate in all simulations. Depending on the mobility ratio (R) between ice–air and ice–ice boundaries grain growth can almost stop.

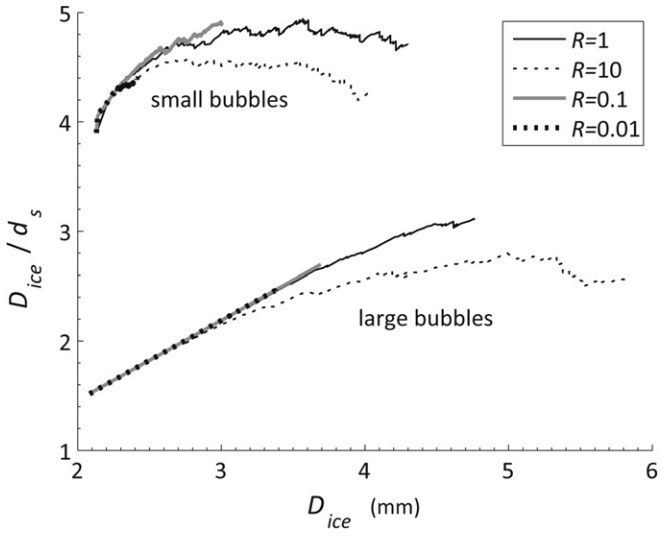


Fig. 5. Graph of mean grain diameter (D_{ice}) divided by mean bubble diameter (d_s) as a function of D_{ice}. Flattening of the curve indicates that the maximum grain size (D_{max}) for the given bubble size and fraction is reached.

gives the results expected from theory and experiments (Glazier et al., 1987; von Neumann, 1952).

The presence of bubbles slows down grain growth significantly, even at the highest ice–air mobility ($R = 10$) (Fig. 3b, c and Fig. 4). At the beginning of each run, many bubble-free grain boundaries exist, especially in model 3 with large bubbles. These boundaries can at first migrate relatively unhindered by bubbles. Initial growth rate is therefore close to that of bubble-free ice, but then quickly decreases as more and more boundaries are slowed down by bubbles. Especially when a triple junction snaps on to a bubble, it tends to remain on it. At low R (<1), grain growth quickly slows down and almost comes to a halt, especially in model 2 with many bubbles. This shows that not only the fraction, but also the size of bubbles is important.

At higher R, grain growth continues, albeit at a slower rate than for bubble-free ice. Since triple junctions mainly remain on bubbles, continued growth is only possible when the bubbles migrate through the material. This occurs since the surface tension of grain boundaries ending on bubbles effectively pulls on these bubbles. Unless the grain boundaries are evenly distributed around the bubble, the net pull on a bubble is non-zero. The bubble can migrate by removal of ice on one side and deposition on the other. This process is visualized well in experiments on grain boundary-induced migration of fluid inclusions by Schmatz et al. (2011). Migration of bubbles leads to merging of bubbles. This has two effects: the bubble size and their size range increases. Numerical simulations by Bons et al. (2004) where spheres were randomly merged, showed that that process eventually results in a power-law distribution of sphere sizes. In our simulations we observe a similar tendency with the development of a few large bubbles and many small ones (Fig. 3b, c).

According to Eq. (2) growth should stop when the maximum grain size (D_{max}) is reached, which is related to the bubble size (d_s) and bubble fraction (f). At this stage, the ratio D_{max}/d_s should remain constant at a fixed f (as is the case in each simulation). This ratio is plotted against grain size in Fig. 5. After an initial increase, D_{max}/d_s settles close to 3 and 4.5 for large and small bubbles, respectively. The different values reflect the different bubble fractions. Slow mobilities follow the same trend as high mobilities, but take much longer to reach a steady-state D_{max}/d_s.

A redistribution and size increase of bubbles (and concomitant reduction of their number) implies a change in microstructure. The microstructure in the bubble-free ice simulation did not change during the simulation. Shape and size distribution at the beginning and the end of the simulation are identical, except for the length scale. For simple scaling reasons, D^2 must increase linearly with time, as it indeed does. This is not the case for growth with bubbles, were the microstructure at the beginning and end of a simulation change significantly where there was significant grain growth (higher R). The growth curves are not straight, but curved: dD^2/dt decreases with time. Assuming that the growth exponent (n in Eq. (1)) remains two, dD^2/dt equals k. Fig. 6 shows the normalised slope of the growth curves (β), using an approximately one year moving window. The normalised growth rate is dD^2/dt divided by that for bubble-free ice (Arena et al., 1997). Owing to the relatively small number of grains, there is much noise. However, a clear tendency for an initial reduction towards a steady-state value of β can be seen. β is reduced by about one order of magnitude at the highest ice-air boundary mobility and by almost three orders of magnitude

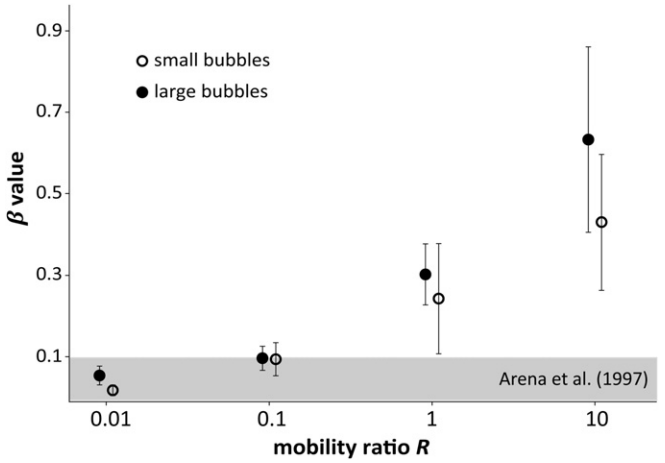


Fig. 6. β values of all simulations as a function of the mobility ratio R. Error bars are one standard deviation. β values reported by Arena et al. (1997) are indicated with the grey area.

Please cite this article in press as: Roessiger, J., et al., Influence of bubbles on grain growth in ice, Journal of Structural Geology (2012), http://dx.doi.org/10.1016/j.jsg.2012.11.003

for the lowest mobility (Fig. 7), which is in the range of β-values reported by Arena et al. (1997) for artificial bubbly ice and glacial ice from Byrd station up to a depth of 279 m.

As bubbles coalesce they do not only increase their mean size (d_s), but also their size and spatial distribution. In the Zener equation (Eq. (2)), this would be reflected in a change in the constant z and the exponent m, which may explain the decreasing grain/bubble size ratio at the end of the $R \geq 1$ simulations (Fig. 5).

4. Discussion

These numerical simulations do not attempt to model any specific natural ice occurrence, but rather serve to provide insight in grain growth behaviour in the presence of a second phase, here ice with bubbles. Results, however, apply equally well to other systems (Hiraga et al., 2010a,2010b; Ohuchi and Nakamura, 2007; Yamazaki et al., 1996). First of all, by varying the relative mobility of ice–air boundaries, the simulations show that this has a first order effect on grain growth. It effectively controls the drag rate of the second phase. At about the same fraction of the second phase, many small bubbles slow down grain growth more than a few large ones, in accordance with the Zener equation (Eq. (2)). However, the bubbles grow by coalescence and thus the maximum grain size (D_{max}) keeps increasing. The trajectory towards a stable grain/bubble size ratio is the same for all mobilities, but proceeds much slower at low bubble mobility.

It should be stressed that, in our simulations, bubbles always stay on grain boundaries and cannot be dropped. Alley et al. (1986b), Weikusat et al. (2009), and others observed bubbles inside grains, which means that bubbles in natural ice can be dropped. However, images of bubbly ice in the upper few hundred metres of ice sheets, show that the vast majority of bubbles are actually on grain boundaries (e.g. Fig. 3 in Arnaud et al., 1998). This is important in determining whether grain boundaries are in the *fast regime* (relatively unhindered by dragging particles or bubbles) or in the *slow regime*, where drag controls their velocity (Hsueh et al., 1982; Urai et al., 1986). Although Alley et al. (1986b) argue that ice recrystallisation is in the fast regime, the microstructures suggest otherwise, at least for the upper hundreds of metres where air bubbles are still relatively large. In our simulations dropping of bubbles would actually rarely occur, even if the code would potentially allow it. This is because most bubbles are on triple or more junctions and even if a triple junction would "break away" from the bubble, the bubble still remains on one or more grain boundaries. Bubbles on a single boundary also do not get dropped, because during static grain growth the boundaries remain close to straight.

The fact that dropped bubbles are observed in natural ice indicates that grain boundary migration is not only driven by surface energy, but also by internal strain energy. This much higher driving force (Urai et al., 1986) would lead to a consistent and faster migration of a boundary, which would allow it to drop a bubble. Although dynamic recrystallisation in the upper few hundred meters of ice sheets is controversial, (Alley, 1992; Duval and Castelnau, 1995; Faria et al., 2002) compelling arguments for it, even above the firn–ice transition, were recently published (Kipfstuhl et al., 2009; see also the review by Faria et al., this issue).

The simulations show three regimes of grain growth. In the first regime, most boundaries are not slowed down by bubbles and grain growth rate approximates that of bubble-free ice ($D << D_{max}$; $\beta \leq 1$). This stage is most noticeable in model 3, where the initial spacing between bubbles is significantly larger than the average grain diameter. As grains grow, their mean diameter approaches that of the bubble spacing and most grain boundaries are hindered in their movement by bubbles. Grain growth is significantly slowed down ($D \approx D_{max}$; $\beta << 1$). Drag by grain boundaries leads to a spatial redistribution of bubbles and merging to a change in the bubble-size distribution. The Zener parameters z and m change gradually. This continues until in the third regime a stable microstructure (spatial and size distribution of bubbles) is reached, after

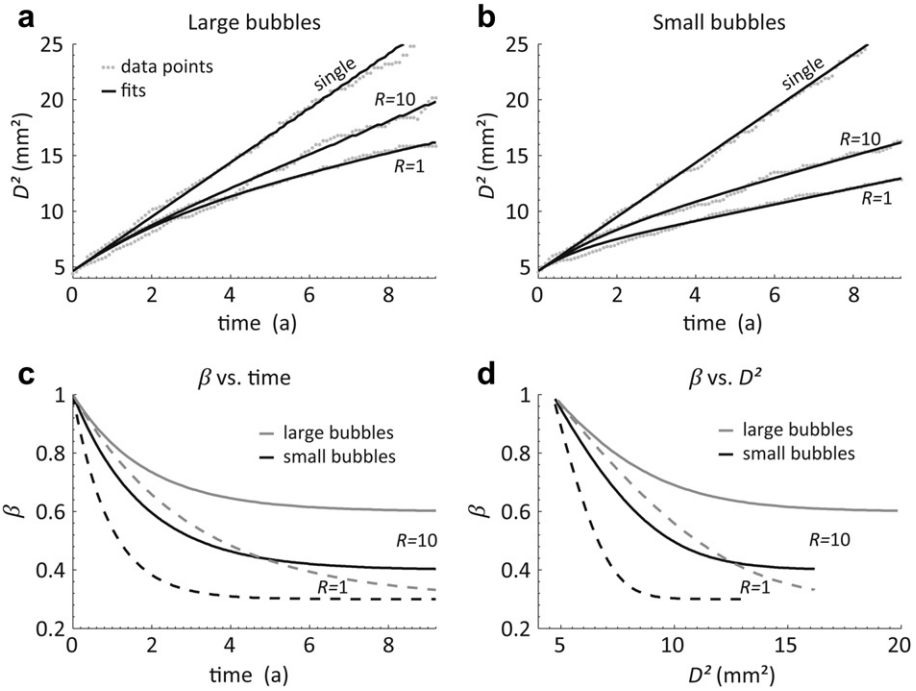


Fig. 7. (a) Squared mean grain diameter versus time graphs for pure ice and ice with large bubbles at $R = 1$ and 10. Grey dots are the simulation results and closed lines the best fit with Eq. (8). (b) Same as (a) for simulations with small bubbles. (c) β versus time curves derived from a best fit of Eq. (8) to the numerical simulations. (d) Same β versus squared mean grain diameter.

Please cite this article in press as: Roessiger, J., et al., Influence of bubbles on grain growth in ice, Journal of Structural Geology (2012), http://dx.doi.org/10.1016/j.jsg.2012.11.003

which grain growth continues, without a change in microstructure, except for length scale. The Zener parameters z and m have reached steady-state values. In our simulations this third regime may have been approached in the $R = 1$ and 10 simulations. Only at this stage would one expect a linear increase of D^2 with time, as the microstructural parameter k_0 does not change any more.

4.1. Exponential model for k

The different regimes of grain growth discussed above suggest that the microstructural parameter β decays from close to unity (regime 1) to a lower steady state value in regime 3. This change reflects the change from the initial microstructure to a steady-state one, achieved after a certain amount of grain growth, and hence time. It is postulated here that the rate of change of the microstructure, and hence of β, depends on how strongly the microstructure deviates from the stable steady-state one. Considering this, it is suggested to express β as follows:

$$\beta_{(t)} = \beta_\infty + (1-\beta_\infty)\exp^{-ct} \tag{7}$$

This means that $\beta_{(t=0)}$ is that for a bubble-free foam texture and decays to $\beta_\infty << 1$ as t goes to infinity (Fig. 5). The rate of microstructure change is given by the "decay constant" c. Combining Eq. (7) with Eq. (1) and integrating gives:

$$D^2_{(t)} = D^2_0 + \frac{k}{c}\left((1-\beta_\infty) + c\beta_\infty t - (1-\beta_\infty)\exp^{-ct}\right) \tag{8}$$

We can fit Eq. (8) to the growth curves from our simulations by varying both β_∞ and c and using the value of k derived from the bubble-free simulation (Fig. 6). For $R = 1$ and $R = 10$ we achieve a reasonable fit. Since little growth occurred in the lower air-ice mobility experiments, a fit in such cases would be meaningless. β_∞ should be a function of the fraction of bubbles and their mobility. More and longer simulations are needed to investigate this function. A first impression is given by Fig. 7, where we see an approximately linear increase in β_∞ with the logarithm of the relative ice-air mobility within the range $0.1 \leq R \leq 10$. This suggests that when $R >> 10$, bubbles have negligible effect on the growth rate ($\beta_\infty \approx 1$), while at $R << 0.1$ β_∞ is so small that growth is effectively inhibited on time scales relevant to nature.

4.2. Implications for the experimental determination of growth exponents of diverse materials

The above results have direct implications for the interpretation of grain growth experiments in two-phase materials (Hiraga et al., 2010a; Ohuchi and Nakamura, 2007; Olgaard and Evans, 1988; Tullis and Yund, 1982; Yamazaki et al., 1996). Here one usually takes several samples from the same starting material, heats these for different periods of time at a constant temperature and then measures the grain size at the end of each experiment. Equation (1) is then fitted to the grain size − time data. If the grain growth mechanism is unknown, this fit has two unknown parameters, k and n. It is rarely taken into account that k is not a constant when the microstructure is not in steady state. Erroneously assuming that k is constant can lead to errors in the growth exponent n and possibly in the inferred growth mechanism that determines n.

Hiraga et al. (2010b), for example, find very high n-values in forsterite + enstatite aggregates, even up to $n > 6$, which they use to infer that grain growth is controlled by grain-boundary diffusion ($n = 4$). It is critical to assess whether the microstructure has reached a steady state in their experiments. Our simulations show that significant growth must have taken place before a steady state (regime 3) has been reached. Fitting Eq. (1) to data from regime 1 and 2 leads to too high apparent grain growth exponents. One can see this, if one fits Eq. (1) to the numerical simulations, assuming k is constant. This would give n-values in the order of 3, which would suggest that the rate controlling mechanism is perhaps Ostwald ripening. This is clearly not the case, as the driving mechanism in our model is known to scale with m^2/s, and hence $n = 2$. We do not wish to imply that grain growth in the example of forsterite + enstatite aggregates is not controlled by grain boundary diffusion. Clearly this is possible. The question, however, is how well constrained this interpretation is in experiments with less than five-fold increase in grain diameter, as is the case in, for example, Hiraga et al. (2010b), where the steady-state regime (with constant k) may just set in. Extreme care should be taken when extrapolating these data from experimental time scales of 50 h to millions of years (Hiraga et al., 2010a).

Even if grain growth is controlled by grain boundary diffusion ($n = 4$) at the grain scale of about one micron as is often used in experiments (Hiraga et al., 2010a; Ohuchi and Nakamura, 2007; Olgaard and Evans, 1988; Tullis and Yund, 1982; Yamazaki et al., 1996), care should be taken in extrapolating the growth law (here with $n = 4$) to larger grain sizes. If second-phase material can be transferred across the grains (e.g. by grain boundary diffusion), the process modelled here (controlled by grain boundary curvature with $n = 2$) would also occur. Neglecting the transition regimes 1 and 2, the growth law for such as material would be an addition of $D^4 \propto k_{gbd} \cdot t$ and $D^2 \propto \beta_\infty k_{\mathrm{curve}} \cdot t$, where k_{gbd} and k_{curve} stand for the growth constants of grain-boundary diffusion and curvature controlled growth, respectively. Rewriting this in the growth rate (dD/dt) as a function of grain size gives:

$$\frac{\mathrm{d}D}{\mathrm{d}t} = \frac{k_{gbd}}{4D^3} + \frac{\beta_\infty k_{\mathrm{curve}}}{2D} \tag{9}$$

At a grain size of $D = \sqrt{k_{gbd}/2\beta_\infty k_{\mathrm{curve}}}$ the two processes contribute equally to grain growth. At a smaller grain size, grain-boundary diffusion dominates, while at a larger grain size grain-boundary curvature dominates. Clearly, experiments carried out in the first regime should not be extrapolated to grain sizes in the second regime. Unfortunately, this means that experiments need to be carried out over much longer time scales than currently done. This is because growth must be long enough to establish a steady state (regime 3) and at larger grain sizes. Since these time scales may be impossible (>>years) for practical reasons, numerical models such as the ones shown here may provide a solution.

5. Conclusions

We presented two-dimensional numerical simulations of two-phase grain growth, where the second phase has a high dihedral angle, as in the case of bubbly ice. Three growth regimes could be identified:

- Regime 1: Relatively fast growth, when bubble spacing is larger than the grain diameter and many ice−ice boundaries can still migrate unhindered by bubbles. The growth parameter k is initially close to that for bubble-free ice, but it rapidly declines.
- Regime 2: Once bubble spacing is in the order of the grain diameter, a transitional regime starts where growth decreases as the spatial and size distribution evolves to a steady state. The decline in k slows down.
- Regime 3: Steady-state growth with a constant microstructure and hence, constant k.

As the growth exponent in the numerical simulations is known to be two, the decline in growth rate is solely due to the decline in

Please cite this article in press as: Roessiger, J., et al., Influence of bubbles on grain growth in ice, Journal of Structural Geology (2012), http://dx.doi.org/10.1016/j.jsg.2012.11.003

the growth rate parameter k, which can be expressed (after Arena et al., 1997) by the factor β, which is the ratio of actual k (a function of microstructure, including bubble volume fraction and size distribution) and the one for bubble-free ice. We propose to express the evolution of β as an exponential function of time.

Our numerical simulations indicate that extreme caution should be taken in the interpretation of two-phase grain growth experiments, if these are in regimes 1 and 2. Failure to recognise growth in these regimes can lead to spurious growth exponents and erroneous interpretations of the growth mechanism.

Acknowledgements

The authors gratefully acknowledge funding from the DFG research project 1776/7 and ESF Research Networking Programme Micro-Dynamics of Ice (Micro-DICE). We thank Marco Herwegh and an anonymous reviewer for very helpful suggestions to improve the manuscript.

References

Alley, R.B., 1992. Flow-law hypotheses for ice-sheet modelling. Journal of Glaciology 38, 245–256.
Alley, R.B., Perepezko, J.H., Bentley, C.R., 1986a. Grain Growth in polar ice: I. Theory. Journal of Glaciology 32, 415–424.
Alley, R.B., Perepezko, J.H., Bentley, C.R., 1986b. Grain Growth in polar ice: II. Application. Journal of Glaciology 32, 425–433.
Anderson, M.P., 1986. Simulation of grain growth in two and three dimensions. In: Hanse, N., Juul Jensen, N., Leffers, D., Ralp, T.B. (Eds.), Annealing Processes – Recovery, Recrystallization and Grain Growth. Risø National Laboratory, Roskilde, pp. 15–34.
Arena, L., Nasello, O.B., Levi, L., 1997. Effect of bubbles on grain growth in ice. Journal of Physical Chemistry B 101, 6109–6112.
Arnaud, L., Barnola, J.M., Duval, P., 2000. Physical modeling of the densification of snow/firn and ice in the upper part of polar ice sheets. In: Hondoh, T. (Ed.), Physics of Ice Core Records. Hokkaido University Press, Sapporo, pp. 285–305.
Arnaud, L., Gay, M., Barnola, J.M., Duval, P., 1998. Imaging of firn and bubbly ice in coaxial reflected light: a new technique for the characterization of these porous media. Journal of Glaciology 44, 326–332.
Azuma, N., Miyakoshi, T., Yokoyama, S., Takata, M., 2012. Impeding effect of air bubbles on normal grain growth of ice. Journal of Structural Geology. http://dx.doi.org/10.1016/j.jsg.2012.05.005.
Barnes, P.R.F., Mulvaney, R., Robinson, K., Wolff, E.W., 2002. Observations of polar ice from the Holocene and the glacial period using the scanning electron microscope. Annals of Glaciology 35, 559–566.
Becker, J.K., Bons, P.D., Jessell, M.W., 2008. A new front-tracking method to model anisotropic grain and phase boundary motion in rocks. Computers & Geosciences 34, 201–212.
Bons, P.D., Arnold, J., Elburg, M.A., Kalda, J., Soesoo, A., van Milligen, B.P., 2004. Melt extraction and accumulation from partially molten rocks. Lithos 78, 25–42.
Bons, P.D., Jessell, M.W., Evans, L., Barr, T.D., Stüwe, K., 2001. Modelling of anisotropic grain growth in minerals. Geological Society of America Memoir 193, 39–49.
Bons, P.D., Koehn, D., Jessell, M.W. (Eds.), 2008. Microdynamic Simulation. Springer, Berlin.
Brodhag, S.H., Herwegh, M., 2010. The effect of different second-phase particle regimes on grain growth in two-phase aggregates: insights from in situ rock analogue experiments. Contributions to Mineralogy and Petrology 160, 219–238.
De Bresser, J.H.P., Ter Heege, J.H., Spiers, C.J., 2001. Grain size reduction by dynamic recrystallization: can it result in major rheological weakening? International Journal of Earth Sciences 90, 28–45.
De La Chapelle, S., Castelnau, O., Lipenkov, V., Duval, P., 1998. Dynamic recrystallization and texture development in ice as revealed by the study of deep ice cores in Antarctica and Greenland. Journal of Geophysical Research 103, 5091–5105.
Durand, G., Weiss, J., Lipenkov, V., Barnola, J.M., Krinner, G., Parrenin, F., Delmonte, B., Ritz, C., Duval, P., Rothlisberger, R., Bigler, M., 2006. Effect of impurities on grain growth in cold ice sheets. Journal of Geophysical Research 111, F01015.
Duval, P., Castelnau, O., 1995. Dynamic recrystallisation of ice in polar ice sheets. Journal De Physique 5, 197–205.
Evans, B., Renner, J., Hirth, G., 2001. A few remarks on the kinetics of static grain growth in rocks. International Journal of Earth Sciences 90, 88–103.
Faria, S.H., Freitag, J., Kipfstuhl, S., 2010. Polar ice structure and the integrity of ice-core paleoclimate records. Quaternary Science Reviews 29, 338–351.
Faria, S.H., Kipfstuhl, S., Azuma, N., Freitag, J., Hamann, I., Murshed, M.M., Kuhs, W.F., 2009. The multiscale structure of Antarctica. Part I: inland ice. In: Hondoh, T. (Ed.), Physics of Ice Core Records II. Hokkaido University Press, Sapporo, pp. 39–59.
Faria, S.H., Ktitarev, D., Hutter, K., 2002. Modelling evolution of anisotropy in fabric and texture of polar ice. Annals of Glaciology 35, 545–551.
Faria, S.H., Weikusat, I., Azuma, N. The microstructure of polar ice. Journal of Structural Geology, in this issue.
Glazier, J.A., Gross, S.P., Stavans, J., 1987. Dynamics of two-dimensional soap froths. Physical Review A 36, 306–312.
Gow, A.J., 1969. On the rate of growth of grains and crystals in south polar firn. Journal of Glaciology 8, 241–252.
Gow, A.J., 1971. Depth-time-temperature relationships of crystal growth in polar glaciers. Cold Regions Research and Engineering Lab Hanover NH Report 300, 20.
Gow, A.J., Meese, D.A., Alley, R.B., Fitzpatrick, J.J., Anandakrishnan, S., Woods, G.A., Elder, B.C., 1997. Physical and structural properties of the Greenland Ice Sheet Project 2 ice core: a review. Journal of Geophysical Research 102, 26559–26575.
Gow, A.J., Williamson, T., 1976. Rheological implications of the internal structure and crystal fabrics of the West Antarctic ice sheet as revealed by deep core drilling at Byrd Station. Geological Society of America Bulletin 87, 1665–1677.
Herwegh, M., Berger, A., 2003. Differences in grain growth of calcite: a field-based modeling approach. Contributions to Mineralogy and Petrology 145, 600–611.
Herwegh, M., Handy, M.R., 1996. The evolution of high-temperature mylonitic microfabrics: evidence from simple shearing of a quartz analogue (norcamphor). Journal of Structural Geology 18, 689–710.
Herwegh, M., Linckens, J., Ebert, A., Berger, A., Brodhag, S.H., 2011. The role of second phases for controlling microstructural evolution in polymineralic rocks: a review. Journal of Structural Geology 33, 1728–1750.
Hiraga, T., Miyazaki, T., Tasaka, M., Yoshida, H., 2010a. Mantle superplasticity and its self-made demise. Nature 468, 1091–1094.
Hiraga, T., Tachibana, C., Ohashi, H., Sano, S., 2010b. Grain growth systematics for forsterite ± enstatite aggregates: effect of lithology on grain size in the upper mantle. Earth and Planetary Science Letters 291, 10–20.
Hondoh, T., 2009. An overview of microphysical processes in ice sheets: toward nanoglaciology. In: Hondoh, T. (Ed.), Physics of Ice Core Records II. Hokkaido University Press, Sapporo, pp. 1–23.
Hsueh, C.H., Evans, A.G., Cobble, R.L., 1982. Microstructure development during final/intermediate stage sintering - I. Pore/grain boundary separation. Acta Metallurgica 30, 1269–1279.
Jessell, M., Bons, P., Evans, L., Barr, T., Stuwe, K., 2001. Elle: the numerical simulation of metamorphic and deformation microstructures. Computers & Geosciences 27, 17–30.
Jessell, M.W., Kostenko, O., Jamtveit, B., 2003. The preservation potential of microstructures during static grain growth. Journal of Metamorphic Geology 21, 481–491.
Ketcham, W.M., Hobbs, P.V., 1969. An experimental determination of the surface energies of ice. Philosophical Magazine 19, 1161–1173.
Kipfstuhl, S., Faria, S.H., Azuma, N., Freitag, J., Hamann, I., Kaufmann, P., Miller, H., Weiler, K., Wilhelms, F., 2009. Evidence of dynamic recrystallization in polar firn. Journal of Geophysical Research 114, B05204.
Lipenkov, V.Y., Salamati, A.N., Duval, P., 1992. Bubbly-ice densification in ice sheets: II. Applications. Journal of Glaciology 43, 398–407.
Manohar, P.A., Ferry, M., Chandra, T., 1998. Five decades of the Zener equation. ISIJ International 38, 913–924.
Mathiesen, J., Ferkinghoff-Borg, J., Jensen, M.H., Levinsen, M., Olesen, P., Dahl-Jensen, D., Svensson, A., 2004. Dynamics of crystal formation in the Greenland NorthGRIP ice core. Journal of Glaciology 50, 325–328.
Montagnat, M., Duval, P., 2000. Rate controlling processes in the creep of polar ice, influence of grain boundary migration associated with recrystallization. Earth and Planetary Science Letters 183, 179–186.
Mullins, W.W., 1989. Estimation of the geometrical rate-constant in idealized 3 dimensional grain-growth. Acta Metallurgica 37, 2979–2984.
Ohuchi, T., Nakamura, M., 2007. Grain growth in the forsterite–diopside system. Physics of the Earth and Planetary Interiors 160, 1–21.
Olgaard, D.L., Evans, B., 1986. Effect of second-phase particles on grain growth in calcite. Journal of the American Ceramic Society 69, C272–C277.
Olgaard, D.L., Evans, B., 1988. Grain growth in synthetic marbles with added mica and water. Contributions to Mineralogy and Petrology 100, 246–260.
Paterson, W.S.B., 1994. The Physics of Glaciers. Pergamon, Oxford.
Petit, J.R., Duval, P., Lorius, C., 1987. Long-term climatic changes indicated by crystal growth in polar ice. Nature 326, 62–64.
Piazolo, S., Jessell, M.W., Bons, P.D., Evans, L., Becker, J.K., 2010. Numerical simulations of microstructures using the Elle platform: a modern research and teaching tool. Journal of the Geological Society of India 75, 110–127.
Piazolo, S., Jessell, M.W., Prior, D.J., Bons, P.D., 2004. The integration of experimental in-situ EBSD observations and numerical simulations: a novel technique of microstructural process analysis. Journal of Microscopy 213, 273–284.
Roessiger, J., Bons, P.D., Griera, A., Jessell, M.W., Evans, L., Montagnat, M., Kipfstuhl, S., Faria, S.H., Weikusat, I., 2011. Competition between grain growth and grain-size reduction in polar ice. Journal of Glaciology 57, 942–948.
Schmatz, J., Schenk, O., Urai, J.L., 2011. The interaction of migrating grain boundaries with fluid inclusions in rock analogues: the effect of wetting angle and fluid inclusion velocity. Contributions to Mineralogy and Petrology 162, 193–208.
Schulson, E.M., Duval, P. (Eds.), 2009. Creep and Fracture of Ice. Cambridge University Press, Cambridge.
Smith, C.S., 1964. Some elementary principles of polycrystalline microstructure. Metallurgical Reviews 9, 1–48.
Stöckhert, B., Duyster, J., 1999. Discontinuous grain growth in recrystallised vein quartz – implications for grain boundary structure, grain boundary mobility, crystallographic preferred orientation, and stress history. Journal of Structural Geology 21, 1477–1490.
Thorsteinsson, T., Kipfstuhl, J., Miller, H., 1997. Textures and fabrics in the GRIP ice core. Journal of Geophysical Research 102, 26583–26599.

Please cite this article in press as: Roessiger, J., et al., Influence of bubbles on grain growth in ice, Journal of Structural Geology (2012), http://dx.doi.org/10.1016/j.jsg.2012.11.003

Tullis, J., Yund, R.A., 1982. Grain growth kinetics of quartz and calcite aggregates. Journal of Geology 90, 301–318.
Urai, J.L., Means, W.D., Lister, G.S., 1986. Dynamic recrystallization of minerals. In: Hobbs, B.E., Heard, H.C. (Eds.), Mineral and Rock Deformation: Laboratory Studies. Geophysical Monograph, pp. 161–200.
von Neumann, J., 1952. Written discussion on grain topology and the relationships to growth kinetics. In: Herring, C. (Ed.), Metallic Materials. American Society for Metals. Metals Park, Cleveland, pp. 108–113.
Weikusat, I., Kipfstuhl, S., Faria, S.H., Azuma, N., Miyamoto, A., 2009. Subgrain boundaries and related microstructural features in EDML (Antarctica) deep ice core. Journal of Glaciology 55, 461–472.
Weygand, D., Bréchet, Y., Lépinoux, J., 1999. Zener Pinning and Grain Growth: a two-dimensional vertex computer simulation. Acta Materialia 47, 961–970.
Yamazaki, D., Kato, T., Ohtani, E., Toriumi, M., 1996. Grain growth rates of $MgSiO_3$ Perovskite and Periclase under lower mantle conditions. Science 274, 2052–2054.

Please cite this article in press as: Roessiger, J., et al., Influence of bubbles on grain growth in ice, Journal of Structural Geology (2012), http://dx.doi.org/10.1016/j.jsg.2012.11.003

APPENDIX 3

MULTISCALE MODELING OF ICE DEFORMATION BEHAVIOR

Multiscale modeling of ice deformation behavior

M. Montagnat[a,*], O. Castelnau[b], P. D. Bons[c], S. H. Faria[d,e], O. Gagliardini[a,i], F. Gillet-Chaulet[a], F. Grennerat[a], A. Griera[f], R. A. Lebensohn[g], H. Moulinec[h], J. Roessiger[c], P. Suquet[h]

[a]*Laboratoire de Glaciologie et Géophysique de l'Environnement, CNRS, UJF - Grenoble I, 38402 Saint-Martin d'Hères, France*
[b]*Procédés et Ingénierie en Mécanique et Matériaux, CNRS, Arts & Métiers ParisTech, 151 Bd de l'hopital, 75013 Paris, France*
[c]*Department of Geosciences, Eberhard Karls University Tübingen, Wilhelmstr. 56, 72074 Tübingen, Germany.*
[d]*Basque Centre for Climate Change (BC3), Alameda Urquijo 4, 48008 Bilbao, Spain*
[e]*Ikerbasque, Basque Foundation for Science, 48011 Bilbao, Spain*
[f]*Departament de Geologia, Universitat Autónoma de Barcelona, 08193 Bellaterra (Cerdanyola del V.), Spain*
[g]*Los Alamos National Laboratory, MS G755, Los Alamos, NM 87545, USA*
[h]*Laboratoire de Mécanique et d'Acoustique, CNRS, UPR 7051, 13402, Marseille cedex 20, France.*
[i]*Institut Universitaire de France (IUF), Paris, France*

Abstract

Understanding the flow of ice in glaciers and polar ice sheets is of increasing relevance in a time of potentially significant climate change. The flow of ice has hitherto received relatively little attention from the structural geological community. This paper aims to provide an overview of methods and results of ice deformation modeling from the single crystal to the polycrystal scale, and beyond to the scale of polar ice sheets. All through these scales, various models have been developed to understand, describe and predict the processes that operate during deformation of ice, with the aim to correctly represent ice rheology and self-induced anisotropy. Most of the modeling tools presented in this paper originate from the material science community, and are currently used and further developed for other materials and environments. We will show that this community has deeply integrated ice as a very useful "model" material to develop and validate approaches in conditions of a highly anisotropic behavior. This review, by no means exhaustive, aims at providing an overview of methods at different scales and levels of complexity.

Keywords: ice mechanical behavior, multiscale modeling, viscoplastic anisotropy, fabric development

Contents

Email address: montagnat@lgge.obs.ujf-grenoble.fr (M. Montagnat)
URL: http://lgge.osug.fr/montagnat-rentier-maurine (M. Montagnat)

1. Introduction

Ice is a common mineral on the Earth's surface, where it occurs as ice Ih. As ice is relatively close to its melting temperature, glaciers and polar ice sheets deform by ductile dislocation creep at strain rates in the order of 10^{-12} to 10^{-6} s^{-1}. Research on the flow of ice is of direct importance to society as it is needed to understand and predict the effects that global warming could have on sea level rise, glacier retreat, etc. There is also an increasing awareness that ice is a valuable analogue for other minerals and crystalline materials, as it is the only common mineral where this creep can be readily observed in nature and in the laboratory. Numerical modeling has become a key method to link the mechanics of ice from the dislocation scale to that of flowing ice masses.

Most of the efforts made to simulate the ductile mechanical behavior of polycrystalline ice are related to the modeling of ice flow and fabric evolution in the conditions of polar ice sheets or glaciers. Ice is increasingly considered a model material to validate micro-macro mechanical approaches for materials with a high viscoplastic anisotropy. Most of the modeling techniques presented in this paper are currently used or further developed for other materials. For geological applications, one main limitation could be related to the "one phase" approach for most of these techniques, well adapted to ice. The reader will find, at the end of the paper, a table summarizing the main aspects of each techniques, with application ranges and limitations.

1.1. Mechanical properties of ductile ice

Ice Ih has an hexagonal crystal structure with a c/a ratio of 1.628. This c/a ratio is very close to the 1.633 value for a closely packed structure, but ice is not closely packed (see Schulson and Duval (2009) for a recent review). The elastic anisotropy of ice single crystals is small. The Young modulus E only varies by about 30%, depending on the direction of the loading axis with respect to the c-axis. The highest value is along the c-axis with $E = 11.8$ GPa at $-16°C$ (Gammon et al., 1983).

Single crystals deform plastically essentially by glide of dislocations on the basal plane. There are three equivalent $< 1\bar{2}10 >$ directions for the Burgers vector, but slip on the basal plane is almost isotropic. In conditions where basal slip is favored, the stress-strain rate relationship after a strain of about 5% can be expressed by a power law with a stress exponent $n = 2 \pm 0.3$ (Higashi et al., 1965; Jones and Glen, 1969; Mellor and Testa, 1969). At similar strain rates, the equivalent stress requested for non-basal slip is about 60 times larger than for basal slip (Duval et al., 1983).

For ice polycrystals deformed under the laboratory conditions (strain rate between about 10^{-8} s^{-1} and 10^{-6} s^{-1} and temperature generally higher than -30°C), strain is essentially due to intracrystalline dislocation glide. The transient creep regime is characterized by a strong directional hardening until the strain-rate minimum is reached for an overall strain of 1% (Duval et al., 1983). This strain-rate decrease can reach three orders of magnitude. It is associated to the development of a strong internal stress field due to plastic incompatibility between grains (Ashby and Duval, 1985; Duval et al., 1983; Castelnau et al., 2008b). A significant part of the transient creep is recoverable, i.e., on unloading a creep specimen, a reverse creep is observed, with reverse strain which can be more than ten times the initial elastic strain (Duval, 1976; Duval et al., 1983). In the secondary creep regime, isotropic polycrystals deform (at similar stress levels) a 100 times slower than a single crystal optimally oriented for basal slip. In this regime, the minimum strain rate and the stress are linked by a power law, referred to as Glen's law in glaciology (Glen, 1955), expressed through a relationship of the form (1) for temperatures lower that -10°C.

$$\dot{\bar{\varepsilon}}_{min} = A\bar{\sigma}^n exp(-E_p/k_B T) \quad (1)$$

with $\bar{\sigma}$ the applied stress, $E_p = 0.72$ eV and the stress exponent $n = 3$ (Barnes et al., 1971; Budd and Jacka, 1989). A is a constant, k_B the Boltzmann constant and T the temperature. Above -10°C, $\dot{\bar{\varepsilon}}_{min}$ rises more rapidly with increasing temperature and cannot be described by this equation (Morgan, 1991). No grain-size effect is expected for power-law secondary creep at laboratory conditions (see Duval and Le Gac (1980); Jacka (1994) for instance). But a grain size effect was, however, measured during transient creep (Duval and Le Gac, 1980).

At strains larger than 1 to 2% (tertiary creep regime), dynamic recrystallization is predominant, and new grain microstructures and crystal orientations are generated (Jacka and Maccagnan, 1984; Duval et al., 2000).

At stresses lower than 0.1 MPa, relevant to deformation conditions in glaciers, ice sheets or planetary bodies, there is a clear indication of a creep regime with a stress exponent lower than two. This indication results from both the analysis of field data and laboratory tests, although the difficulty of obtaining reliable data at strain rates lower than $10^{-10}s^{-1}$ is at the origin of contradictory results (Mellor and Testa, 1969; Barnes et al., 1971; Dahl-Jensen and Gundestrup, 1987; Pimienta et al., 1987; Lipenkov et al., 1997; Goldsby and Kohlstedt, 1997). In particular, Goldsby and Kohlstedt (1997) suggest a grain-size dependence of the ice viscosity associated with this low stress regime, based on laboratory experiments performed on very small grain-size samples. This grain-size effect would be associated with a grain boundary-sliding dominated creep. Its extrapolation to polar ice-core deformation conditions remains controversial (Duval and Montagnat, 2002). Diffusional creep, commonly associated with such conditions in many materials yields a viscosity much higher than that deduced from field data (Lliboutry and Duval, 1985). For a review on ice behavior, see (Duval et al., 2010).

Ice as a model material exhibits a challenging viscoplastic anisotropy owing to the presence of only two independent easy slip systems for the dislocations (basal plane). While five independent systems are required to accommodate an arbitrary deformation in a single crystal (Taylor, 1938), Hutchinson (1977)

showed that four systems are required for allowing an hexagonal polycrystal such as ice to deform. Being able to represent and to take into account this anisotropy in micro-macro models which aim at linking the single crystal scale to the polycrystal scale, is of primary interest to the material science community. This anisotropy needs to be accounted for at the dislocation scale in order to build physically-based model for the activation of (poorly known) secondary slip systems. The impact of dislocation induced internal stress fields, but also the characterization and development of highly heterogeneous strain and stress fields within polycrystals, and their impact on fabric development turn out to be of strong importance (Castelnau et al., 1996a; de la Chapelle et al., 1998).

During gravity-driven flow of glaciers and ice sheets, the macroscopic behavior of ice becomes progressively anisotropic with the development of fabrics (or textures, c-axis preferred orientations). This anisotropy and its development depends on the flow conditions, but strongly influences the response of ice layers to imposed stress (see Gundestrup and Hansen (1984); Van der Veen and Whillans (1990); Mangeney et al. (1997) for pioneer field work and modeling on the subject). Indeed, a polycrystal of ice with most of its c-axes oriented in the same direction deforms at least ten times faster than an isotropic polycrystal, when sheared parallel to the basal planes.
Fabrics basically develop as the result of lattice rotation by intracrystalline slip (Azuma and Higashi, 1985; Alley, 1988, 1992). Dynamic recrystallization can have a major impact on fabric development, especially at temperatures above -10°C close to bedrocks or within temperate glaciers (Alley, 1992; Duval and Castelnau, 1995; de la Chapelle et al., 1998; Montagnat et al., 2009), see Section 5. Questions, however, remain to what extent different recrystallization processes operate as a function of depth in polar ice sheets (Kipfstuhl et al., 2006, 2009; Weikusat et al., 2009).

1.2. Main objectives

Accurate modeling of ice flow under natural conditions is relevant for many scientific objectives, such as the response of ice sheet to climate changes (Seddik et al., 2012), the interpretation of climate signals extracted from ice cores (Faria et al., 2010), the energy balance in extraterrestrial satellites (Sotin et al., 2009), and since a few years, the accurate prediction of sea-level rise that is linked to the behavior of fast-moving coastal glaciers (Gillet and Durand, 2010). In this context, challenges are mainly (i) to establish an ice flow law adapted to low stress conditions, changes in temperatures and impurity content, (ii) to consider the macroscopic anisotropy due to fabric development at the given conditions, (iii) to be able to integrate processes such as dynamic recrystallization that can strongly influence fabric development and the flow law.

The aim of this paper is to present a general overview of the main modeling techniques adapted to ice, and the main modeling results obtained from the single crystal scale to the large scale that is relevant to ice sheet flow modeling. Techniques are highly diverse, from dislocation dynamics (micron scale) to Finite Element methods that are adapted to the whole ice sheet (km scale), via mean-field and full-field micro-macro approaches and coupling with a microstructure evolution models (cm to m scale, limited to a 2D configuration, see 5.2). We will mostly focus on recent advances and topics that are still under development.

2. Modeling ice single crystal behavior

Owing to its high viscoplastic anisotropy, with dislocations gliding mostly on the basal plane, studying and modeling ice single crystal behavior is a challenge for regular approaches.
Recent efforts focused on three main objectives; (i) understanding, representing and taking into account the dislocation dynamics, (ii) improving our knowledge about secondary slip systems in ice, (iii) providing an accurate crystal plasticity constitutive law that can be implemented in mean-field and full-field approaches for micro-macro polycrystal models. For the two first objectives, Dislocation Dynamic models (DD) were used at the scale of the interaction between dislocation populations (Section 2.1). At a larger scale, the Field Dislocation Mechanic modeling approach (FDM) was applied to ice to evaluate the role of internal stresses associated with dislocation fields and arrangements (Section 2.2). Section 2.3 presents a crystal-plasticity model adapted to the transient creep behavior of ice single crystals.

2.1. Dislocation Dynamics modeling

Dislocation dynamics in ice was shown to be scale free and intermittent, thanks to dislocation avalanche measurements via acoustic emissions (Weiss and Grasso, 1997; Weiss et al., 2001; Weiss and Marsan, 2003; Weiss and Montagnat, 2007). Ice was used as a model material for the following reasons: (i) transparency allows direct verification that acoustic emission activity is not related to microcraking, (ii) with the range of stress and temperature considered, diffusion creep is not a significant mechanism, and deformation occurs by dislocation glide only. DD modeling tools were used to better understand and characterize this scale free and intermittent behavior (for example Miguel et al. (2001); Weiss and Miguel (2004)).
Miguel et al. (2001) made use of a discrete dislocation dynamics model with a two-dimensional cross-section of the crystal. This 2D space is randomly filled with edge dislocations gliding along a single slip direction parallel to their respective Burgers vector. This simplification is an effective way to describe materials like ice crystals owing to their strong plastic anisotropy with a single slip system dominating. A basic feature common to most DD models is that dislocations interact with each other through the long-range elastic stress field they produce in the host material. In (Miguel et al., 2001), dislocation velocity depends linearly on this effective stress, and the Peierls stress is set to zero. Mechanisms for dislocation annihilation and multiplication are classically taken into account.

Within this simplified scheme the authors found that dislocations generate a slowly evolving configuration landscape which coexists with rapid collective rearrangements. These arrangements involve a comparatively small fraction of dislocations

and lead to an intermittent behavior of the net plastic response. The model was therefore able to reproduce the fact that dislocations themselves, through the various structures such as dipoles and walls, generate a pinning force landscape that is virtually frozen into a slow state. Creation and annihilation mechanisms allow the system to jump between slow dynamics states through bursts of activity.

More recently, Chevy et al. (2007, 2012) used DD simulations to analyze torsion tests performed on ice single crystals. The tests were performed with the ice-crystal c-axis oriented parallel to the torsion axis so that basal screw dislocations were mainly activated. With synchrotron topography analyses of the deformed samples, it was possible to show that dislocation arrangements were highly heterogeneous, with a scale-invariant character and long-range correlations (Montagnat et al., 2006; Weiss and Montagnat, 2007; Chevy et al., 2010). Although these tests were performed in a way that highly favored basal glide, the double-cross slip mechanisms was invoked to explain this scale invariant dislocation arrangement.

Three-dimensional DD simulations, based on the TRIDIS code (Verdier et al., 1998), were adapted to these torsion tests on ice and the hexagonal structure. Screw dislocation sources were positioned within one slip plane at the periphery of a cylinder submitted to a constant torque. Cross-slip on prismatic planes was made possible thanks to the internal stress induced by the pile-up of basal dislocations in the center of the cylinder (where $\sigma_{app} = 0$), which produces the out-of-plane component needed (see Fig. 1). Simulation results allowed to test this hypothesis, and explain the power law relationship between stress and strain rate (Chevy et al., 2012).

2.2. Field Dislocation Mechanics (FDM)

Field dislocation theory is a mesoscale approach, which aims at taking into account the inhomogeneous distribution of dislocations in plasticity modeling. Therefore, FDM modeling makes it possible to represent and consider the internal stress field created by the dislocation arrangements within the crystal. FDM is a continuous approach able to deal simultaneously with long-range correlations associated with distortion fields, internal stresses due to dislocation arrangements, and short-range correlations (Acharya, 2001). The reader is referred to (Acharya and Roy, 2006; Varadhan et al., 2006; Fressengeas, 2010) for details.

The first application to ice samples was performed in the configuration of the torsion test presented in part 2.1 (Taupin et al., 2007). This test is by itself highly heterogeneous, and this heterogeneity was shown to induce unexpected non-basal slip. Taking into account the coupled dynamics of geometrically necessary screw dislocations gliding in the basal plane (also called "excess" dislocations) and statistical dislocations developed through cross slip in prismatic planes, the model was able to reproduce the creep curves during torsion, and the size effect measured experimentally (see Fig. 2). More recently, the model was used to reproduce the complex scale-invariant character of dislocation arrangements forming during torsion tests on ice single crystals (Chevy et al., 2010). In particular, the fact that the model takes into account both the long-range elastic interactions due to the presence of dislocations and the short-range interactions inherent to the transport of dislocations (obstacles, cross-slip, etc.) allowed to reproduce the shift in control of the dislocation distribution by long-range correlations at low strain to a control by short-range correlations at strain as high as 50%. It was shown that non-basal dislocations activated by the internal stress fields induce a screening potential at large strain, through obstacles such as twist sub-boundaries. However, this screening was shown to be too small to hinder creep acceleration prevailing during torsion creep test on ice single crystals (Chevy et al., 2010).

2.3. Crystal plasticity modeling

Constitutive relations to describe the transient creep of ice single crystals have been proposed by Castelnau et al. (2008b) and then used in a modified version in Suquet et al. (2011). One of the difficulty here is the description of the softening of basal slip in the transient regime, as discussed above. As is usual in crystal plasticity at infinitesimal strains, the strain tensor is decomposed into the sum of an elastic $\boldsymbol{\varepsilon}^{\mathsf{e}}$ and a viscoplastic $\boldsymbol{\varepsilon}^{\mathsf{vp}}$ part

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^{\mathsf{e}} + \boldsymbol{\varepsilon}^{\mathsf{vp}} \, . \tag{2}$$

The elastic strain is related to the local stress tensor $\boldsymbol{\sigma}$ with the local compliance tensor $\mathbf{S}$, and the viscoplastic strain results from slips on a total of M different slip systems:

$$\boldsymbol{\varepsilon}^{\mathsf{e}} = \boldsymbol{S} : \boldsymbol{\sigma} \, , \quad \boldsymbol{\varepsilon}^{\mathsf{vp}} = \sum_{k=1}^{M} \gamma^{(k)} \boldsymbol{\mu}^{(k)} \, . \tag{3}$$

Here, $\boldsymbol{\mu}^{(k)} = \frac{1}{2}(\mathbf{n}^{(k)} \otimes \mathbf{b}^{(k)} + \mathbf{b}^{(k)} \otimes \mathbf{n}^{(k)})$ is the (purely geometric) Schmid tensor depending on the orientation of the slip system (k), $\mathbf{n}$ being the slip plane normal and $\mathbf{b}$ the slip direction (parallel to the Burgers vector and orthogonal to $\mathbf{n}$) in that plane, with $\otimes$ the dyadic product.

Ice crystals, which have an hexagonal symmetry, deform easily by shear on the basal plane, on the three systems $\{0001\} < 11\bar{2}0 >$ which provide only two independent systems. The three prismatic systems $\{1\bar{1}00\} < 11\bar{2}0 >$ provide two more independent systems. An additional independent slip system is thus required to attain any isochoric deformation at the single crystal level and this is achieved by adding the six $< c + a >$ pyramidal systems $\{11\bar{2}2\} < 11\bar{2}\bar{3} >$. In total, $M = 12$ slip systems are taken into account in the present analysis.

In the constitutive relations originally proposed by Castelnau et al. (2008b), the slip rate on the k-th system is related to the resolved shear stress $\tau^{(k)}$ on that system through:

$$\dot{\gamma}^{(k)} = \dot{\gamma}_0^{(k)} \left(\frac{\left|\tau^{(k)}\right|}{\tau_0^{(k)}} \right)^{n^{(k)}} \operatorname{sgn}(\tau^{(k)}), \quad \tau^{(k)} = \boldsymbol{\sigma} : \boldsymbol{\mu}^{(k)}, \tag{4}$$

where $\tau_0^{(k)}$, the reference resolved shear stress on system k, depends on the activity of the other systems through:

$$\dot{\tau}_0^{(k)} = \sum_{\ell=1}^{M} H^{(k,\ell)} \left(\frac{\tau_{sta}^{(\ell)} - \tau_0^{(\ell)}}{\tau_{sta}^{(\ell)} - \tau_{ini}^{(\ell)}} \right) \left| \dot{\gamma}^{(\ell)} \right| . \tag{5}$$

The two material parameters $\tau_{ini}^{(\ell)}$ and $\tau_{sta}^{(\ell)}$ refer, respectively, to the initial value of $\tau_0^{(\ell)}$ at the onset of plasticity (when the $\gamma^{(k)}$'s are small) and to the stationary value of $\tau_0^{(\ell)}$ at saturation when the plasticity is fully developed (*i.e.* when the $\gamma^{(k)}$'s are large). Therefore the contribution of system ℓ in the hardening (or softening) of system k vanishes when $\tau_0^{(\ell)}$ is close to $\tau_{sta}^{(\ell)}$. The hardening matrix $H^{(k,\ell)}$ expresses the influence of the plastic activity of system ℓ on the hardening of system k and is taken to be symmetric. Material data for this model are given in Castelnau et al. (2008b).

In (Suquet et al., 2011), Eqs (4) and (5) are improved in two ways:

1. Kinematic hardening is introduced in (4) through a back stress $X^{(k)}$:

$$\dot{\gamma}^{(k)} = \dot{\gamma}_0^{(k)} \left(\frac{\left| \tau^{(k)} - X^{(k)} \right|}{\tau_0^{(k)}} \right)^{n^{(k)}} \operatorname{sgn}\left(\tau^{(k)} - X^{(k)} \right), \qquad (6)$$

   where the back stress evolves with the plastic activity according to an Armstrong-Frederick type law (Chaboche, 2008):

$$\dot{X}^{(k)} = c^{(k)} \dot{\gamma}^{(k)} - d^{(k)} X^{(k)} \left| \dot{\gamma}^{(k)} \right| - e^{(k)} X^{(k)}, \qquad (7)$$

   including static recovery through coefficient $e^{(k)}$. The introduction of a back stress on each slip system is motivated by the experimental observation of recovery strain developing in single crystals when specimens are subjected to recovery tests (see Section 2.3.2 and Fig. 4).
2. The equation governing the reference resolved shear stress $\tau_0^{(k)}$ is modified into

$$\dot{\tau}_0^{(k)} = \left(\tau_{sta}^{(k)} - \tau_0^{(k)} \right) \dot{p}^{(k)}, \quad \dot{p}^{(k)} = \sum_{\ell=1}^{M} H^{(k,\ell)} \left| \dot{\gamma}^{(\ell)} \right| . \qquad (8)$$

   The motivation for the change in the evolution rule for the reference resolved shear stresses $\tau_0^{(k)}$ is that with the original rule (5) they never reach their stationary value, unless all systems do so at the same time, a condition which cannot be met in a polycrystal (see details in (Suquet et al., 2011)). By contrast, the law (8) ensures convergence of $\tau_0^{(k)}$ towards its stationary value, provided all coefficients $H^{(k,\ell)}$ are positive. Indeed, in this case, $\dot{p}^{(k)}$ is always positive and $p^{(k)}$ is increasing with time, acting on system k in a similar way as the classical cumulated plastic strain of von Mises plasticity. The differential Eq. (8) can be integrated into

$$\tau_0^{(k)}(p^{(k)}) = \tau_{sta}^{(k)} + (\tau_{ini}^{(k)} - \tau_{sta}^{(k)}) \exp(-p^{(k)}), \qquad (9)$$

   which shows that $\tau_0^{(k)} - \tau_{sta}^{(k)}$ has the same sign as $\tau_{ini}^{(k)} - \tau_{sta}^{(k)}$. Furthermore $\tau_0^{(k)}$ tends to $\tau_{sta}^{(k)}$ when $p^{(k)}$ becomes large.

2.3.1. Data for elasticity

As mentioned in Section 1, ice crystals exhibit a low elastic anisotropy, the largest stiffness ($E \sim 11.8$GPa) being along the c-axis (Fig. 3). The tensor of elastic moduli (in Kelvin's notations) at $-16°C$ is given by (10) (Gammon et al., 1983),

$$\begin{pmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sqrt{2}\sigma_{23} \\ \sqrt{2}\sigma_{13} \\ \sqrt{2}\sigma_{12} \end{pmatrix} = \begin{pmatrix} 13930. & 7082. & 5765. & 0. & 0. & 0. \\ 7082. & 13930. & 5765. & 0. & 0. & 0. \\ 5765. & 5765. & 15010. & 0. & 0. & 0. \\ 0. & 0. & 0. & 6028. & 0. & 0. \\ 0. & 0. & 0. & 0. & 6028. & 0. \\ 0. & 0. & 0. & 0. & 0. & 6848. \end{pmatrix} \begin{pmatrix} \varepsilon_{11}^{e} \\ \varepsilon_{22}^{e} \\ \varepsilon_{33}^{e} \\ \sqrt{2}\varepsilon_{23}^{e} \\ \sqrt{2}\varepsilon_{13}^{e} \\ \sqrt{2}\varepsilon_{12}^{e} \end{pmatrix}, \qquad (10)$$

where all entries are in MPa and 3 is the axis of transverse isotropy (c-axis of the hexagonal crystalline structure). For conditions prevailing in ice sheets and glaciers, elastic constants vary little with temperature: a temperature change of 5°C only modifies the elastic constants by about 1.5%.

2.3.2. Data for basal slip

The literature provides a number of experimental data for the behavior of ice crystals deformed in such a way that only basal slip is activated. Due to the very large viscoplastic anisotropy of ice single crystals, it is stressed that mechanical tests have to be carried out very carefully to avoid any heterogeneity of the stress field within the specimen (Boehler et al., 1987).

Mechanical tests on single crystals where solely non-basal systems are activated have not been reported so far. This would require straining the crystal along or perpendicular to the c-axis, but unfortunately any unavoidable deviation from perfect alignment activates basal slip. Duval et al. (1983) has given upper bounds for the flow stress on non-basal systems.

Consequently, only the material parameters of Eq. (6) relevant for basal slip can be identified with confidence from experimental data on single crystals :

- First, data compiled by Duval et al. (1983) were used to determine the stationary flow stress and the stress-sensitivity exponent $n^{(k)}$ of basal slip. There is quite a large spread in these experimental results from different authors. Despite these uncertainties, the stress-sensitivity exponent for basal slip can be directly identified from these experimental data (numerical values are reported in Table 1), whereas the stationary flow stress depends on both the stationary reference stress $\tau_{sta}^{(k)}$ and the stationary backstress $X^{(k)}$.
- Next, data from Weertman (1973) were used for the identification of the transient creep regime of basal systems. Single crystals were deformed under uniaxial compression at different strain rates, with c-axis oriented at 45° from the loading direction (Fig. 4). The observed stress peak is associated with the increase in density of mobile dislocations (Duval et al., 1983), a behavior typical for material with very low initial dislocation density (see Sauter and Leclercq (2003); Cochard et al. (2010)). These tests shed light on the softening of basal slip in the transient regime. The static recovery term $e^{(k)}$ in the constitutive law (7) helps achieving the correct stationary stress at very small strain rates (since $X^{(k)}$ tends to a constant value $c^{(k)}/d^{(k)}$ at large shear $\gamma^{(k)}$ if static recovery is not introduced).
- Finally, the recovery test of Taupin et al. (2008) performed on single crystals under uniaxial compression was consid-

ered. Here the c-axis orientation was not precisely defined experimentally, but it made an angle "less than 10°" with the compression direction. The single crystal was submitted to four creep loadings for 30 minutes separated by unloading stages for respectively 1 minute, 10 minutes, and 100 minutes (Fig. 4). Upon reloading, the strain rate is larger than just before the last unloading, indicating that dislocations are rearranging during the time intervals where the specimen is unloaded. This is accounted for in the model by the back stress $X^{(k)}$, and by $e^{(k)}$.

Fig. 4 shows the good match between the model (constitutive Eq. (6)) with the set of parameters given in Table 1 and these experimental results.

3. Mean field approaches for the mechanical response of ice polycrystals

3.1. Microstructure characterization

From the mechanical point of view, polycrystalline materials have to be considered as a specific class of composites. They are composed of many grains, with grain size in the range of mm to cm for natural ice. Grains are assembled in a random way, *i.e.* their size, shape, and lattice orientation do generally not depend on the size, shape, and orientation of the surrounding grains (Fig. 5). Therefore, the microstructure of ice polycrystals can hardly be described exactly in 3-D, unless one makes use of tomography techniques (Rolland du Roscoat et al., 2011). From (2-D) thin sections, one can at best access a statistical characterization of the 3-D grain arrangement *e.g.* with the help of cross-correlation functions, although the description is generally limited to a few parameters, such as the average grain size and grain shape (aspect ratio). In the Euler orientation space, microstructure description is based on the distribution of crystal lattice orientations (Orientation Distribution Function, ODF, or crystallographic texture, often denoted "fabric" in the geophysical community). The complex behavior of polycrystalline materials comes from the anisotropic behavior at the grain scale, closely related to the symmetry of the crystal lattice. This is true for all quantities of interest here, such as elasticity, viscoplasticity and thermal dilation. Grains with different lattice orientations react differently to a given stress level. As far as grain boundaries maintain the cohesion of the material, the *local* stress (*i.e.* inside a grain) differs from the *overall* one (the applied stress), leading to a heterogeneous distribution of stress and strain fields within the polycrystal.

Most research efforts in the past years have focussed on the understanding of the build-up of these heterogeneities, in relation with the microstructure and local (grain) behavior, since they greatly influence the overall behavior (for ice, see Grennerat et al. (2012) for instance). For instance, plasticity in a polycrystal can start far below the macroscopic yield stress, as it is sufficient that the local stress reaches the local yield stress somewhere in the structure where stress concentration is large enough, such as along grain boundaries (Brenner et al., 2009).

There are basically two strategies to get the mechanical response: mean-field (this section) and full-field (next section) approaches. For both of them, the key issue is the estimation of the stress or strain localization (or heterogeneities), in relation to the microstructure and local behavior of grains. Basically, the problem to be solved is to find an equilibrated stress field, related to a compatible strain field with the local constitutive relation, both fields fulfilling the applied boundary conditions. In the following, we review (not in an exhaustive way) some homogenization techniques used for the investigation of the mechanical behavior of ice polycrystals.

3.2. Linear thermo-elasticity

For reasons that will become evident below, let us consider the case of thermo-elastic ice polycrystals. The local constitutive relation at point $(\mathbf{x})$ reads

$$\boldsymbol{\varepsilon}(\mathbf{x}) = \mathbf{S}(\mathbf{x}) : \boldsymbol{\sigma}(\mathbf{x}) + \boldsymbol{\varepsilon}_0(\mathbf{x}), \tag{11}$$

with $\boldsymbol{\varepsilon}_0$ a stress-free thermal strain (*e.g.* a dilation), due to temperature changes. The local stress $\boldsymbol{\sigma}(\mathbf{x})$ can be related to the overall stress (applied at the polycrystal scale) by means of the stress-concentration tensor $\mathbf{B}(\mathbf{x})$ for the purely elastic problem

$$\boldsymbol{\sigma}(\mathbf{x}) = \mathbf{B}(\mathbf{x}) : \bar{\boldsymbol{\sigma}} + \boldsymbol{\sigma}_{\text{res}}(\mathbf{x}), \tag{12}$$

with $\boldsymbol{\sigma}_{\text{res}}$ the residual stress, *i.e.* the stress field remaining locally when the overall load is suppressed ($\bar{\boldsymbol{\sigma}} = \mathbf{0}$). It can be shown that the overall polycrystal behavior takes a similar form as Eq. (11)

$$\bar{\boldsymbol{\varepsilon}} = \tilde{\mathbf{S}} : \bar{\boldsymbol{\sigma}} + \tilde{\boldsymbol{\varepsilon}}_0 \ , \tag{13}$$

with symbols $\tilde{.}$ and $\bar{.}$ denoting the homogenized (or effective) property and the volume average over the whole polycrystal volume (also denoted $\langle . \rangle$), respectively. Therefore, one has $\bar{\boldsymbol{\sigma}} = \langle \boldsymbol{\sigma}(\mathbf{x}) \rangle$ and $\bar{\boldsymbol{\varepsilon}} = \langle \boldsymbol{\varepsilon}(\mathbf{x}) \rangle$, and it can be shown that the effective compliance $\tilde{\mathbf{S}}$ and the effective thermal strain $\tilde{\boldsymbol{\varepsilon}}_0$ are given by (Laws, 1973)

$$\tilde{\mathbf{S}} = \langle \mathbf{S}(\mathbf{x}) : \mathbf{B}(\mathbf{x}) \rangle, \quad \tilde{\boldsymbol{\varepsilon}}_0 = \langle \boldsymbol{\varepsilon}_0(\mathbf{x}) : \mathbf{B}(\mathbf{x}) \rangle \ . \tag{14}$$

Since, for thermo-elastic polycrystals, the elastic compliance and the thermal dilation coefficients are uniform properties inside grains, the quantities $\mathbf{S}(\mathbf{x})$ and $\boldsymbol{\varepsilon}_0(\mathbf{x})$ in Eq. (11) can be replaced by the corresponding homogeneous values $\mathbf{S}^{(r)}$ and $\boldsymbol{\varepsilon}_0^{(r)}$ of the considered phase (r). A similar substitution can be made in Eq. (14), leading to

$$\tilde{\mathbf{S}} = \sum_r c^{(r)} \mathbf{S}^{(r)} : \bar{\mathbf{B}}^{(r)} \ , \quad \tilde{\boldsymbol{\varepsilon}}_0 = \sum_r c^{(r)} \boldsymbol{\varepsilon}_0^{(r)} : \bar{\mathbf{B}}^{(r)} \tag{15}$$

with $\bar{.}^{(r)}$ indicating the average over the volume of phase (r), *e.g.* $\bar{\mathbf{B}}^{(r)} = \langle \mathbf{B}(\mathbf{x}) \rangle^{(r)}$, and $c^{(r)}$ the volume fraction of phase (r). Here, a *mechanical phase* (r) denotes the set of all grains of the polycrystal having the same crystal orientation; those grains have different shape and environment but their elastic and thermal properties are identical. From (15), it can be observed that the sole knowledge of the mean (phase average) values $\bar{\mathbf{B}}^{(r)}$ is sufficient to estimate the overall polycrystal behavior. It can be anticipated that, if the quantities $\bar{\mathbf{B}}^{(r)}$ can be calculated without having to know the complete field of $\mathbf{B}(\mathbf{x})$, computation will be

way faster. Hence the name of "mean-field" approaches presented here.

With the effective behavior (Eq. 14) in hand, statistical averages over crystal orientations (r) can be estimated. Basically, two quantities can be obtained from mean-field approaches:

1. The phase average stress (or first moment) $\bar{\boldsymbol{\sigma}}^{(r)} = \langle \boldsymbol{\sigma}(\mathbf{x}) \rangle^{(r)}$

$$\bar{\boldsymbol{\sigma}}^{(r)} = \bar{\mathbf{B}}^{(r)} : \bar{\boldsymbol{\sigma}} + \bar{\boldsymbol{\sigma}}^{(r)}_{\text{res}}, \tag{16}$$

   with $\bar{\boldsymbol{\sigma}}^{(r)}_{\text{res}}$ the average residual stress of phase (r). The knowledge of $\bar{\boldsymbol{\sigma}}^{(r)}$ for all phases (r) allows investigating the so-called *interphase* heterogeneities, *i.e.* the variation of the phase average stress with respect to the crystal orientation.
2. Deeper insight into the stress distribution can be obtained from the second moment $\langle \boldsymbol{\sigma} \otimes \boldsymbol{\sigma} \rangle^{(r)}$ of the stress.This second moment can be obtained by a derivation of the effective energy with respect to local compliances, see (Bobeth and Diener, 1987; Kreher, 1990; Ponte-Castañeda and Suquet, 1998; Brenner et al., 2004).

The standard deviation of the stress distribution within a given crystal orientation (r) can be estimated from these two moments as the square root of $< \boldsymbol{\sigma} \otimes \boldsymbol{\sigma} >^{(r)} - < \boldsymbol{\sigma} >^{(r)} \otimes < \boldsymbol{\sigma} >^{(r)}$ (*i.e.* the mean of the square of the stress minus the square of the mean); it is related to the width of the stress distribution in crystal orientation (r), and accounts for both the heterogeneity of stress distribution *inside* grains but also for the heterogeneity *between* grains of identical orientation but exhibiting different shapes and having different neighborhood. Similar relations can be derived for the strain statistics.

3.3. Reuss and Voigt approximations

First, two very basic models can be derived, namely *Reuss* (also called *static* in the viscoplastic context) and *Voigt* (or *Taylor*) models. The Reuss model is constructed by considering a uniform stress throughout the polycrystal, *i.e.* $\boldsymbol{\sigma}(\mathbf{x}) = \bar{\boldsymbol{\sigma}} \ \ \forall \mathbf{x}$, or equivalently $\mathbf{B}(\mathbf{x}) = \mathbf{I}$ (with $\mathbf{I}$ the identity tensor), and leads to vanishing intra– and inter–granular stress heterogeneities, and uniform strain within grains. The Voigt model considers uniform strain, *i.e.* $\boldsymbol{\varepsilon}(\mathbf{x}) = \bar{\boldsymbol{\varepsilon}} \ \ \forall \mathbf{x}$, *i.e.* no intra– and inter–granular strain heterogeneities, and uniform stress within grains. These models violate strain compatibility and stress equilibrium, respectively, and are of limited accuracy when the local behavior is highly nonlinear and/or highly anisotropic, as will be illustrated in the next section. Besides simplicity, the main interest of Reuss and Voigt models is based on their bounding character, since they provide, respectively, a lower and an upper bound for the effective stress potential.

3.4. The Self-Consistent (SC) scheme

Unlike full-field approaches detailed in the Section 4, mean-field methods are based on a statistical description of the microstructure, *e.g.* based on few n-points correlation functions, so that the exact position and shape of a specific grain with respect to its neighbors is not known. However, as already introduced, all grains exhibiting the same crystallographic orientation are treated as a single mechanical phase. Owing to the random character of the microstructure with all grains playing geometrically similar roles, the Self-Consistent (SC) scheme (Hershey, 1954; Kröner, 1958; Willis, 1981) is especially well suited for polycrystals. This model, which provides a relatively simple expression for $\bar{\mathbf{B}}^{(r)}$, relies on specific microstructures exhibiting perfect disorder and infinite size graduation (Kröner, 1978). The SC scheme has often been described as if the interaction between each grain and its surrounding could be approximated by the interaction between one ellipsoidal grain with the same lattice orientation as the original grain and a homogeneous equivalent medium whose behavior represents that of the polycrystal, taking thus advantage of the analytical solution of Eshelby (1957) for the inclusion/matrix interaction. This reasoning led to the conclusion that the SC scheme implicitly considers uniform stress and strain rate inside grains. This interpretation turns out to be incorrect, since intraphase stress and strain heterogeneities do not vanish as explained above, see Ponte-Castañeda and Suquet (1998) for a review.

The ability of the SC scheme to estimate polycrystal behavior is shown in Fig. 6. Numerical reference solutions from the full-field FFT method (see Section 4) have been generated for many randomly generated Voronoi microstructures, and ensemble average over these random microstructures has been calculated in order to attain results that are representative for a Representative Volume Element, *i.e.* a volume sufficiently large to be statistically representative of the material (Kanit et al., 2003; Lebensohn et al., 2004b). In Fig. 6, we provide results for the effective behavior, that is entirely defined by the effective reference stress $\tilde{\sigma}_0$ which enters in the effective constitutive relation

$$\frac{\dot{\bar{\varepsilon}}_{eq}}{\dot{\varepsilon}_0} = \frac{\bar{\sigma}'_{eq}}{\tilde{\sigma}_0} \tag{17}$$

with $\dot{\varepsilon}_0$ a reference strain rate (taken here equal to $\dot{\gamma}_0$), and $\bar{\sigma}'_{eq}$ and $\dot{\bar{\varepsilon}}_{eq}$ the effective equivalent stress and strain rate respectively ($\bar{\sigma}'_{eq} = \sqrt{3\bar{\boldsymbol{\sigma}} : \bar{\boldsymbol{\sigma}}/2}$, $\dot{\bar{\varepsilon}}_{eq} = \sqrt{2\dot{\bar{\boldsymbol{\varepsilon}}} : \dot{\bar{\boldsymbol{\varepsilon}}}/3}$). Calculations are performed for various viscoplastic anisotropy contrasts (or slip system contrasts) at the grain level, defined by the ratio between non-basal and basal reference shear stresses, *i.e.* $\tau_0^{(Pr)}/\tau_0^{(Ba)}$ for $\tau_0^{(Pyr)} = \tau_0^{(Pr)}$. It can be observed that the SC model perfectly reproduces the reference full-field (FFT) results. Note also that the Reuss bound, often used for highly anisotropic materials like ice, predicts a much too soft overall behavior. This simple approach does not allow to make a realistic link between local and overall rheologies. We also report in this figure the standard deviations (or overall heterogeneities) of equivalent stress and strain rate. These standard deviations have been calculated over the whole polycrystal. Recall that they account for both intra- and inter-granular field heterogeneities for both SC and FFT approaches. It can be observed that the increase of standard deviation with the slip system contrast is well reproduced by the SC scheme, although some discrepancies with FFT results arise at very large contrasts (mostly for the strain-rate fluctuation). Note again that Reuss and Voigt bound do not reproduce these results, even in a qualitative way, since they predict, by construction, vanishing fluctuation of stress and strain rate, respectively. Unlike these simple approaches, the SC scheme

not only predicts the correct effective stress, but also accurately captures the field heterogeneities within the polycrystal. Similar agreement have been obtained for Voronoi and EBSD 2-D microstructures under antiplane shear by Lebensohn et al. (2005).

3.5. Nonlinear viscoplasticity

The mean-field estimate of *nonlinear materials* is significantly more complex than the thermo-elastic case treated above. We consider the case of a viscoplastic polycrystal of ice in which grains are deforming by glide of dislocations on specific slip planes, as discussed above, with slip rates given by Eq. (4), so that the local strain rate reads, since elastic deformations are neglected:

$$\dot{\boldsymbol{\varepsilon}}(\mathbf{x}) = \sum_k \boldsymbol{\mu}_{(k)}^{(r)} \dot{\gamma}_{(k)}(\mathbf{x}) \; . \tag{18}$$

Here, reference stresses τ_0 and stress sensitivities n are supposed to be constant. The constitutive Eq. (18) can also be written

$$\dot{\boldsymbol{\varepsilon}}(\mathbf{x}) = \mathbf{M}(\mathbf{x}) : \boldsymbol{\sigma}(\mathbf{x}) \tag{19}$$

with

$$\mathbf{M}(\mathbf{x}) = \sum_k \frac{\dot{\gamma}_0^{(k)}}{\tau_0^{(k)}} \left| \frac{\boldsymbol{\mu}^{(k)}(\mathbf{x}) : \boldsymbol{\sigma}(\mathbf{x})}{\tau_0^{(k)}} \right|^{n^{(k)}-1} \boldsymbol{\mu}^{(k)}(\mathbf{x}) \otimes \boldsymbol{\mu}^{(k)}(\mathbf{x}) \; . \tag{20}$$

Obviously, the viscous compliance $\mathbf{M}$ relating $\dot{\boldsymbol{\varepsilon}}(\mathbf{x})$ and $\boldsymbol{\sigma}(\mathbf{x})$ – which plays a similar role as $\mathbf{S}$ in Eq. (11) – is not uniform within a phase, owing to the stress sensitivities $n \neq 1$ and the heterogeneity of $\boldsymbol{\sigma}$ in the phases. Consequently, (14) cannot be replaced by (15) for nonlinear behavior. The basic method to deal with such nonlinear behavior is to define a *Linear Comparison Polycrystal* (LCP) having the same microstructure as the real nonlinear polycrystal, and to which the linear homogenization scheme applies (Ponte-Castañeda and Suquet, 1998). Of course, the effective behavior estimated this way remains nonlinear, since the definition of the LCP depends on the applied macroscopic stress. The difficult part of the problem consists of finding the best *linearization* procedure leading to the optimal selection of the LCP. Since decades, there has been quite a number of propositions in the literature dealing with this issue, leading to a *generalization* of the SC scheme for nonlinear behavior. The local constitutive relation given by Eqs (18-20) has to be linearized in a suitable way to obtain a form similar to (11), with $\mathbf{S}$ and $\dot{\boldsymbol{\varepsilon}}_0$ uniform per phase (and where $\boldsymbol{\varepsilon}$ is replaced everywhere by $\dot{\boldsymbol{\varepsilon}}$). Generally speaking, the linearization can be expressed in the form depicted in Fig. 7, (Liu and Ponte Castañeda, 2004).

$$\dot{\gamma}_{(k)}(\mathbf{x}) = \alpha_{(k)}^{(r)} \tau_{(k)}(\mathbf{x}) + \dot{e}_{(k)}^{(r)}, \tag{21}$$

thus leading to the following expressions for $\mathbf{S}^{(r)}$ and $\dot{\boldsymbol{\varepsilon}}_0^{(r)}$

$$\mathbf{S}^{(r)} = \sum_k \alpha_{(k)}^{(r)} \boldsymbol{\mu}_{(k)}^{(r)} \otimes \boldsymbol{\mu}_{(k)}^{(r)} \; , \qquad \dot{\boldsymbol{\varepsilon}}_0^{(r)} = \sum_k \dot{e}_{(k)}^{(r)} \boldsymbol{\mu}_{(k)}^{(r)} \; , \tag{22}$$

where the shear compliance $\alpha_{(k)}^{(r)}$ and stress-free slip-rate $\dot{e}_{(k)}^{(r)}$ can be easily expressed with respect to two reference shear stresses $\check{\tau}_{(k)}^{(r)}$ and $\hat{\tau}_{(k)}^{(r)}$, see Fig. 7. The optimal choice (from the point of view of the variational mechanical problem) of those reference stresses is not straightforward; this is the main reason why several extensions of the SC scheme have been proposed in the literature. Obviously, all of them reduce to the same SC model in the linear case $n = 1$.

Following Ponte Castañeda (1996), Masson et al. (2000) proposed the so-called "affine" (AFF) linearization scheme which is based on the simple idea of a linear behavior (21) tangent to the nonlinear one (4) at the mean shear stress, leading to

$$\check{\tau}_{(k)}^{(r)} = \hat{\tau}_{(k)}^{(r)} = \langle \tau_{(k)} \rangle^{(r)}, \quad \alpha_{(k)}^{(r)} = \left. \frac{\partial \dot{\gamma}}{\partial \tau} \right|_{\tau = \check{\tau}_{(k)}^{(r)}} . \tag{23}$$

The main limitations of this procedure are discussed in detail in Masson et al. (2000) and Bornert and Ponte Castañeda (1998). One of them is the violation of rigorous upper bounds for the effective behavior. More generally, the affine extension is known to overestimate the overall viscosity, *i.e.* to predict an effective behavior that is too stiff. This negative feature can be alleviated by means of the energy formulation originally proposed by Ponte Castañeda (1996) (see Bornert et al. (2001)).

Alternative, more sophisticated ways to generalize the SC scheme have been proposed by Ponte Castañeda and coworkers during the last decades. The basic idea of these methods is to guide the choice of the properties of the LCP by a suitably designed variational principle. An "optimal" solution has been obtained in the context of the so-called "variational" procedure (VAR) (Ponte Castañeda, 1991), which was extended to polycrystals by De Botton and Ponte Castañeda (1995), leading to the choice

$$\check{\tau}_{(k)}^{(r)} = 0, \quad \hat{\tau}_{(k)}^{(r)} = \left[< \tau_{(k)}^2 >^{(r)} \right]^{1/2} \; . \tag{24}$$

The main advantage of this procedure is to provide a rigorous bound, sharper than the Voigt bound, for the effective potential. More recently, the "second-order" (SO) method of Ponte Castañeda (2002), extended to polycrystals in (Liu and Ponte Castañeda, 2004) has been proposed. It leads to reference shear stresses reading

$$\check{\tau}_{(k)}^{(r)} = \langle \tau_{(k)} \rangle^{(r)}, \quad \hat{\tau}_{(k)}^{(r)} = \check{\tau}_{(k)}^{(r)} \pm \left[< (\tau_{(k)} - \check{\tau}_{(k)}^{(r)})^2 >^{(r)} \right]^{1/2} \; . \tag{25}$$

The main differences between AFF, VAR, and SO models may be summarized as follows. The AFF estimate can be regarded as a relatively simple model, allowing rapid computations which can even be rather accurate for polycrystals with weak grain anisotropy and small stress sensitivity. However, its predictions can become unrealistic (*e.g.* bound violation) at a strong anisotropy or nonlinearity. Contrary to AFF, for which linearization only accounts for the phase average stress, VAR accounts for the second moments of the stress, whereas the SO procedure accounts for both the phase average stress *and* intraphase standard deviation (first and second moments) to build the LCP. They can therefore provide better estimates in cases of highly heterogeneous stress distributions, such as for strongly nonlinear or anisotropic polycrystals. Applications of the VAR

procedure to polycrystals with grains having cubic or hexagonal crystallographic structures can be found in (Nebozhyn et al., 2001; Liu et al., 2003).

Finally, the "tangent" (TGT) extension of the SC scheme (Molinari et al., 1987; Lebensohn and Tomé, 1993), often referred to as the "VPSC model" in the literature, is based on the same tangent linearization (23) as the AFF method. However, unlike the AFF extension, this procedure takes advantage of the fact that, for power-law polycrystals with a single stress exponent n for all slip systems, the tangent behavior (21) can be replaced by a secant-like relation, with $\dot{\epsilon}^{(r)}_{(k)} = 0$ and $\alpha^{(r)}_{(k)}$ replaced by $\alpha^{(r)}_{(k)}/n$. The same procedure is further applied at the macroscopic level, leading to an inconsistent definition for the stress localization tensor $\mathbf{B}^{(r)}$ that combines a secant description for the local and global behaviors but a tangent analysis for the inclusion/matrix interaction (Masson et al., 2000). When expressed in the form of tangent expressions, it can be shown that $\dot{\tilde{\boldsymbol{\varepsilon}}}_0$ differs from the exact relation given in (15).

3.5.1. Application to natural ices: effective behavior

Application of homogenization techniques to natural ices aims at understanding (and predicting) the anisotropic behavior of strongly textured specimens, as encountered at depth in natural ice sheets. As will be seen in section 6, the viscoplastic anisotropy of polycrystals significantly influences ice flow at large scales (Mangeney et al., 1996; Gillet-Chaulet et al., 2006; Pettit et al., 2007; Martín et al., 2009). Castelnau et al. (1998) reported mechanical tests performed on specimens from the GRIP ice core (Central Greenland). Along the ice core, the ice microstructure, and in particular the crystallographic fabric, is evolving; with increasing depth, randomly oriented c-axis at the surface of the ice sheet tend to concentrate towards the *in situ* vertical direction down to a depth of $\sim$ 2600 m. Beneath this depth, less pronounced textures are observed due to the initiation of migration recrystallization (Thorsteinsson et al., 1997). In (Castelnau et al., 1998), the experimental stationary creep behavior of those ices have been obtained for two loading conditions (Fig. 8). The first one corresponds to an *in situ* vertical compression, showing an increasing flow stress (decreasing strain rate for a constant applied stress) with increasing depth, since the activation of non-basal slip systems is necessary for pronounced fabrics. The second loading condition corresponds to *in situ* horizontal shear, promoting basal slip and resulting in a softening of the ice with increasing fabric strength. It can be seen that for a given applied stress, strain rates can vary by more than two orders of magnitude depending on the orientation of the applied stress with respect to the specimen fabric, reflecting the very strong viscoplastic anisotropic of ice specimens.

The effective behavior predicted by the affine (AFF) SC model is compared to the experimental data in Fig. 8. It can be observed that the agreement is excellent, meaning that the relation between fabric and effective rheology is very well captured by the model. The model captures correclty the increasing anisotropy from the surface down to $\sim$ 2600 m depth, and the decrease below. The difference by more than two orders of magnitude between the vertical and shear strain-rates at $\sim$ 2600 m is also well reproduced, although this was a challenging feature for the model. To get these results, the reference shear stress $\tau^{(k)}_0$ entering the local constitutive relation, and also the stress sensitivity $n^{(k)}$, for each slip system (k), had to be identified from comparison with a database that included single-crystal experimental tests, and polycrystal ones on many different crystallographic textures (Castelnau et al., 2008b). The resulting single-crystal rheology, used as input in the SC model to get the effective behavior described above, is shown in Fig. 9. For basal slip, agreement with experimental data from the literature is almost perfect. Non-basal systems are much stiffer than the basal systems, and pyramidal slip is found to be much more difficult than prismatic slip. These results are in good agreement with the available data on single crystals, and in qualitative agreement with the known dislocation structure in ice. Therefore, it can be anticipated that the affine SC model does a good job in making the link between the grain and the polycrystal scales, and provides an accurate estimate of the mechanical interaction between deforming grains. In other words, one can anticipate that results shown in Fig. 8 are based on a realistic description of the mechanical interaction between grains and physical deformation processes (dislocation glide) at the (sub)grain level. It can also be seen on Fig. 9 that this identification procedure leads to different stress sensitivities for the different slip system families. A value $n^{(k)} = 2$ was imposed for basal slip in accordance with experimental data, but values for prismatic and pyramidal systems were considered as adjustable parameters. It is also worth noting that the affine model perfectly reproduces an effective stress sensitivity (i.e. at the polycrystal scale) $\tilde{n} = 3$ in agreement with experimental data, although the two major slip systems, basal and prismatic slip, have stress sensitivities smaller than 3 ($n^{(bas)} = 2.0$, $n^{(pr)} = 2.85$). A larger value was considered only for pyramidal slip ($n^{(py)} = 4.0$), but it is worth mentioning that the contribution of pyramidal slip is only very minor (<2%). It can be concluded that, in ice, although basal slip is by far the most active deformation mechanism, secondary slip systems are of great importance for explaining the polycrystal behavior. Basal slip alone does not allow for plastic deformation of ice polycrystals, since it only provides two independent slip systems. Secondary slip systems, here prismatic and pyramidal slip, *must* therefore be activated to add two more independent slip systems. The strength of these stiffer mechanisms determines the viscoplastic anisotropy at the grain scale, and therefore they also control the level of inter– and intra–granular heterogeneities of stress and strain(-rate), and therefore the effective polycrystal rheology. Similar conclusions have been drawn for olivine, a mineral with only three independent slip systems (Castelnau et al., 2008a, 2009, 2010a,b). We therefore anticipate that the strong effect of secondary deformation mechanisms observed here might be a general feature for all polycrystalline materials with less than four independent slip systems. The corollary of these results is that simple or *ad hoc* polycrystal models, such as the Reuss (uniform stress) model, in which ice polycrystals can deform with only basal slip, cannot be accurate. This has been shown for example in (Castelnau et al., 1997): whatever the strength used for prismatic and pyramidal systems, the Reuss model is

not able to reproduce the very large anisotropy of GRIP specimens shown in Fig. 8. This comes from the fact that internal stresses, that have a large influence on the material behavior, are ignored.

Finally, it is also worth mentioning that the TGT SC approach, used in earlier studies, *e.g.* (Castelnau et al., 1997), does not provide as good a match to experimental data as the AFF SC extension. There can be two reasons for that: (i) first of all, it is now known that the inconsistency in the formulation of the TGT SC version leads to an underestimation of the internal stress level, predicting a too soft polycrystal behavior (Gilormini, 1995; Masson et al., 2000); (ii) second, by construction, the TGT model is limited to grain behavior for which all slip systems exhibit the same stress sensitivity $n^{(k)}$. When applied to ice, one must thus consider $n^{(k)} = 3.0$ for all systems, including basal slip, in order to get an effective $\tilde{n} = 3$. The fact that the AFF extension does not have this limitation might also explain a better consistency with experimental data.

3.5.2. Application to natural ices: texture development

Using the Reuss approximation, Van der Veen and Whillans (1994) and Castelnau and Duval (1994) described the fabric evolution under compression, tension, simple and pure shear. Van der Veen and Whillans (1994) needed to impose a kind of "recrystallization" criterion (see Section 5.1) to be able to correctly represent the single-maximum fabric (with c-axis oriented along one direction) in ice deforming in pure shear. Nevertheless, the Reuss approximation faces inconsistency to describe the fabric evolution at the polycrystal scale, as it requires additional kinematical constraints to link the grain rotation-rate with the polycrystal rotation-rate. In most of the "Reuss" type models, these two rates are supposed to be equal, although the velocity field is not continuous.

Models that modify this homogeneous stress assumption were proposed by Azuma (1994) and Thorsteinsson (2002). They introduce some redistribution of stress through neighborhood interaction to define the crystal strain at a given bulk equivalent strain. In particular, Thorsteinsson (2002) defines a crystal arrangement on a three-dimensional cubic grid, where each crystal has six nearest neighbors. The nearest neighbor interaction (NNI) is taken into account by defining a local softness parameter for each crystal which modifies the stress acting on the central crystal compared to the macroscopic stress. This softness parameter further influences the rotation rate of the crystal lattice compared to the bulk. For uniaxial compression tests, the fabrics obtained with the NNI formulation are less concentrated than the ones where no NNI is considered. The reason for this is that the NNI formulation allows all crystals to deform to some extent, while only "soft" crystals would deform in the no-NNI formulation. The fabric obtained after 50% shortening strain compares qualitatively well with the one measured along the GRIP ice core at a depth where the strain is similar (1293 m) (Thorsteinsson et al., 1997).

The VPSC model in its "tangent" version was applied to simulate the fabric development along ice cores (Castelnau et al., 1996b,a, 1998). In (Castelnau et al., 1996a), a comparison was made with bound estimates (Reuss and Voigt). Fabrics simulated in uniaxial compression and extension were found to be qualitatively similar for all models. However, large differences in the rate of fabric development were found. This was explained by the different interaction stiffness between grain and matrix for the three approaches. The fabrics obtained with the VPSC model for uniaxial deformation were in close agreement with the one measured along the ice core (see Fig. 10). In particular, this model well reproduced the fabric evolution along the GRIP ice core within the upper 650 m where dynamic recrystallization is not supposed to strongly impact this evolution (Castelnau et al., 1996b). Lower down, the modeled fabric concentration is too high. Although Castelnau et al. (1996b) attributed this discrepancy to the effects of rotation recrystallization along the core, it was later shown that the tangent approximation overestimates the lattice rotation.In simple shear, the single-maximum fabric found along the ice cores or experimentally could not be reproduced with the VPSC scheme. To get close to this fabric, an extensive (and probably unrealistic) activity of non-basal slip systems was required. More recently, the "second order" (SO) mean field method of Ponte Castañeda (2002) was used to simulate the fabric development along the Talos Dome ice core (Montagnat et al., 2012). Although no recrystallization mechanisms were implemented in this version, the fabric development was astonishingly well reproduced, under the crude assumption of uniaxial compression with a constant strain rate (see Fig. 11). In particular, a good match was obtained when the initial fabric is non isotropic and similar to the one measured in the top firn, at 18 m depth. The cumulated compressive strain along the core was derived from the thinning function provided by the TALDICE-1 chronology (Buiron et al., 2011). The good prediction performed by the VP-SO scheme is probably due to the fact that this SO approach provides a better estimate of the effective behavior than the classical tangent "VPSC" model does in the case of strongly anisotropic materials such as ice (see Section 3.5). Nevertheless, the modeled fabric evolution could not capture the strengthening rate associated with the Glacial to Interglacial climatic transition. At these transition, a change in ice viscosity is expected. It induces an higher sensitivity to the impact of shear stress increasing with depth, that the modeling approach did not considered.

It is also recalled that the heterogeneity of shear on slip systems at the grain level gives rise to heterogeneities of lattice rotation, and therefore generates intragranular misorientations that somehow spread crystal orientations. It is however worth mentioning that *all* models presented above do not consider this strain heterogeneity for estimating fabric evolutions at finite strain. Even in VAR and SO procedures, intraphase strain heterogeneities are considered for defining the LCP, but so far not for estimating microstructure evolutions. As a consequence, mean-field approaches generally predict too sharp textures. The same applies to the prediction of strain hardening, associated with dislocation processes such as storage and annealing. A quantitative study, based on comparisons with reference results obtained by a FFT full-field approach, can be found in (Castelnau et al., 2006).

Most of the efforts to simulate the fabric development in ice, and especially along ice cores, had to face the fact that recrystallization mechanisms could impact this fabric development. This was, most of the time, the analysis made for the observed discrepancies between simulated and measured fabrics (Van der Veen and Whillans, 1994; Wenk et al., 1997; Castelnau et al., 1996b; Thorsteinsson, 2002). Some efforts to implement recrystallization mechanisms in mean-field approaches will be described in Section 5.1.

3.6. Modeling the elasto-viscoplastic behavior

Transient creep is typically encountered when ice flow changes direction, such as in glaciers flowing above irregular bedrock or submitted to tide forcing close to the sea-shore or in icy satellites. During laboratory experiments, transient creep is characterized by a strain-rate drop of more than two orders of magnitude before reaching the secondary creep close to 1% strain, following Andrade's law (Duval, 1978). This decrease is associated with the development of large internal stress fields due to intergranular interactions and a strong kinematic hardening (Duval et al., 1983; Ashby and Duval, 1985; Castelnau et al., 2008b). To reproduce this transient behavior, one has to consider the coupling between elasticity and viscoplasticity that gives rise to the so-called "long-term memory effect", as explained below.

The application of homogenization schemes to the *elasto-viscoplasticity* of polycrystals is more complicated than for viscoplasticity, see for instance (Laws and McLaughlin, 1978). In short, it can be shown that, even in the simple case of a polycrystal comprising grains whose behavior exhibits a single relaxation time (so-called "short-term memory"), the effective behavior exhibits a continuous spectrum of relaxation time ("long-term memory effect") (Sanchez-Hubert and Sanchez-Palencia, 1978; Suquet, 1987). In other words, the overall behavior of a polycrystal is not of the Maxwell type (parallel association of a spring and a dashpot with constant viscosity), even though the individual grains do exhibit local Maxwell type behavior. The basic difference between elasto-viscoplasticity and viscoplasticity is that, for elasto-viscoplasticity, the local strain rate depends on both the stress (viscous part) and the stress-rate (elastic part), whereas it only depends on the stress for viscoplasticity. Therefore, the local strain rate not only depends on the actual local stress, but also on the whole stress history from the initial specimen loading at $t = 0$ up to the current time. To obtain the exact effective mechanical response at time t, it is thus required to keep track of all information (or internal variables) corresponding to the strains at all previous times, and therefore the problem is not simple. Within mean-field approaches, some approximations (with hopefully limited effects on the accuracy of the solution) are thus necessary.

Basically, two approaches have been proposed to deal with this issue. A promising method based on an incremental variational procedure has been proposed by Lahellec and Suquet (2006, 2007). These authors have shown that the homogenization of a *linear visco-elastic* material (*i.e.* with $n = 1$) can be expressed in terms of a homogenization problem for a linear thermoelastic composite with non piecewise uniform eigenstrains.One advantage of this formulation is that it can make use of the intraphase heterogeneities of stress and strain (-rate), and it can therefore probably provide accurate results even at high stress sensitivity and/or local anisotropy. An alternative approach, which provides a good compromise between accuracy of the solution and simplicity of the formalism, is the so-called "affine" Self-Consistent method of Masson and Zaoui (1999). It is based on the correspondence principle (Mandel, 1966), which states that the elasto-viscoplastic problem can be reduced to a simpler homogenization problem (in fact similar to a standard thermo-elastic problem) if solved in Laplace space. One difficulty of this approach is the calculation of the inverse Laplace transforms, that has to be carried out numerically. An approximate inversion procedure, adapted for creep, has been proposed by Brenner et al. (2002b). It has provided promising results for the creep behavior of Zirconium alloys (Letouzé et al., 2002; Brenner et al., 2002a), since it retains the long-term memory effect associated with the elasto-viscoplastic coupling. Recent developments (Ricaud and Masson, 2009) have shown that an internal variable formulation arises naturally from this affine method, providing results in perfect match with reference FFT solutions in the case of linear viscoelasticity (Vu et al., 2012).

To the best of our knowledge, this affine method is the only mean-field approach that has been applied to simulate the transient creep of ice (Castelnau et al., 2008b). Applications make use of the crystal plasticity model for single crystals detailed in Section 2.3. It was shown that the strong hardening *amplitude* during the transient creep (*i.e.* the decrease of the overall strain rate by several orders of magnitude) is explained by the stress redistribution within the specimen: when the overall stress is applied instantaneously, the instantaneous polycrystal response is purely elastic, and since the elastic anisotropy is small, stress distribution within and between grains is almost uniform. But plastic deformation comes into play rapidly to cause a strong redistribution of stress (with large interphase and intraphase heterogeneities) due to the strong viscoplastic anisotropy at the grain scale. This significantly reduces the overall strain rate. On the other hand, the experimental hardening rate (*i.e.* the time necessary to reach the secondary creep regime) is much too slow to be explained by the same process, and is attributed to the hardening of hard-glide slip systems (prismatic slip) in the transient regime, associated with dislocation processes (Fig. 12).

4. Full field approaches for the polycrystal

Mean-field approaches have been extensively used to predict the mechanical behavior of ice polycrystals, and the fabric development as measured along ice cores. Due to its high viscoplastic anisotropy, deformation in ice is expected to be strongly heterogeneous, with a strong impact of grain interactions and kinematic hardening (Duval et al., 1983; Hamman et al., 2007; Montagnat et al., 2011; Grennerat et al., 2012). The mean-field approaches described above are based on the statistical characterization of the intragranular mechanical fields (in terms of average grain stresses and strain rates, and, in the

most advanced formulations, also through the determination of the intracrystalline average field fluctuations), but the actual micromechanical fields remain inaccessible to these homogenization approaches. Modeling the full intracrystalline heterogeneity that develops in ice polycrystals requires the use of full-field approaches. This part will concentrate on full-field approaches that are using the Fast Fourier Transform method to solve the constitutive equations in a discretized polycrystal. It aims at studying the correlation between the heterogeneous deformation patterns that appear inside the constituent single-crystal grains of an ice aggregate and their corresponding crystallographic orientations, along with the influence of other factors, such as orientation and size of neighboring grains. Both viscoplastic and elasto-viscoplastic behavior were investigated, and are presented in the two following sections.

4.1. Viscoplastic approach - FFT

4.1.1. Viscoplastic FFT-based formulation

The intracrystalline states that are developed during creep of polycrystalline ice can be obtained using an extension of an iterative method based on FFT, originally proposed by Moulinec and Suquet (1998) and Michel et al. (2001) for linear and nonlinear composites (Lebensohn et al., 2009; Montagnat et al., 2011). This formulation was later adapted to polycrystals and applied to the prediction of texture development of fcc materials (Lebensohn, 2001), and in turn used for the computation of field statistics and effective properties of power-law 2D polycrystals (Lebensohn et al., 2004a, 2005) and 3D cubic, hexagonal (Lebensohn et al., 2004b) and orthorhombic (Castelnau et al., 2008a) materials. The FFT-based formulation was also applied to compute the development of local misorientations in polycrystalline copper, with direct input from orientation images (Lebensohn et al., 2008). As will be detailed is Section 4.2 it was further extended to transient behavior with an elasto-viscoplastic formulation (Idiart et al., 2006; Suquet et al., 2011; Lebensohn et al., 2012). The FFT-based full-field formulation for viscoplastic polycrystals is conceived for a periodic unit cell, provides an exact solution of the governing equations, and has better numerical performance than a FE calculation for the same purpose and resolution. The viscoplastic FFT-based formulation consists in finding a strain-rate field, associated with a kinematically-admissible velocity field, which minimizes the average of local work-rate, under the compatibility and equilibrium constraints. The method is based on the fact that the local mechanical response of a periodic heterogeneous medium can be calculated as a convolution integral between the Green function of a linear reference homogeneous medium and the actual heterogeneity field. Such type of integrals reduce to a simple product in Fourier space, therefore the FFT algorithm can be used to transform the heterogeneity field into Fourier space and, in turn, to get the mechanical fields by antitransforming that product back to real space. However, since the actual heterogeneity field depends precisely on the a priori unknown mechanical fields, an iterative scheme should be implemented to obtain, upon convergence, a compatible strain-rate field and a stress field in equilibrium.

The periodic unit cell representing the polycrystal is discretized by means of a regular grid $\{x^d\}$, which in turn determines a corresponding grid of the same dimensions in Fourier space $\{\xi^d\}$. Velocities and tractions along the boundary of the unit cell are left undetermined under the sole condition of periodicity. An average velocity gradient $V_{i,j}$ is imposed to the unit cell, which gives an average strain rate $\bar{\dot{\varepsilon}}_{ij} = \frac{1}{2}(V_{i,j} + V_{j,i})$. The local strain-rate field is a function of the local velocity field, i.e. $\dot{\varepsilon}_{ij}(\mathbf{v}_k(\boldsymbol{x}))$, and can be split into its average and a fluctuation term: $\dot{\varepsilon}_{ij}(\mathbf{v}_k(\boldsymbol{x})) = \bar{\dot{\varepsilon}}_{ij} + \tilde{\dot{\varepsilon}}_{ij}(\tilde{\mathbf{v}}_k(\boldsymbol{x}))$, where $\mathbf{v}_i(\boldsymbol{x}) = \bar{\dot{\varepsilon}}_{ij}x_j + \tilde{\mathbf{v}}_i(\boldsymbol{x})$. By imposing periodic boundary conditions, the velocity fluctuation field $\tilde{\mathbf{v}}_k(\boldsymbol{x})$ is assumed to be periodic across the boundary of the unit cell, while the traction field is antiperiodic, to meet equilibrium on the boundary between contiguous unit cells. The local constitutive equation that relates the deviatoric stress $\boldsymbol{\sigma}'(\boldsymbol{x})$ and the strain rate $\dot{\boldsymbol{\varepsilon}}(\boldsymbol{x})$ at point $\boldsymbol{x}$ is obtained from Eqs (18) to (20).

If $p(\boldsymbol{x})$ is the unknown pressure field introduced by the incompressibility constraint, the Cauchy stress field can be written as:

$$\boldsymbol{\sigma}(\boldsymbol{x}) = \boldsymbol{L}^0 : \dot{\boldsymbol{\varepsilon}}(\boldsymbol{x}) + \boldsymbol{\varphi}(\boldsymbol{x}) - p(\boldsymbol{x})\boldsymbol{I} \tag{26}$$

where the polarization field $\boldsymbol{\varphi}(\boldsymbol{x})$ is given by:

$$\boldsymbol{\varphi}(\boldsymbol{x}) = \boldsymbol{\sigma}'(\boldsymbol{x}) - \boldsymbol{L}^0 : \dot{\boldsymbol{\varepsilon}}(\boldsymbol{x}) \tag{27}$$

where $\boldsymbol{L}^0$ is the stiffness (viscosity) of a linear reference medium. Eqs. (26) and (27) amount to transform the actual heterogeneity problem into an equivalent one, corresponding to a homogenous medium with eigen-strain-rates. Note, however, that the above defined polarization field depends on the unknown $\dot{\boldsymbol{\varepsilon}}(\boldsymbol{x})$. Combining Eq. (27) with the equilibrium and the incompressibity conditions gives:

$$L^0_{ijkl}v_{k,lj}(\boldsymbol{x}) + \boldsymbol{\varphi}_{ij,j}(\boldsymbol{x}) - p_{,i}(\boldsymbol{x}) = 0,\ v_{k,k}(\boldsymbol{x}) = 0 \tag{28}$$

Assuming for a moment that the polarization field $\boldsymbol{\varphi}(\boldsymbol{x})$ is known, the system of partial differential equations (28), with periodic boundary conditions across the unit cell boundary, can be solved by means of the Green function method.
If G_{km} and H_m are the periodic Green functions associated with the velocity and hydrostatic pressure fields, the solutions of system (28) are convolution integrals between those Green functions and the actual polarization term. The velocity gradient, after some manipulation is given by:

$$\tilde{v}_{i,j}(\boldsymbol{x}) = \int_{R^3} G_{ik,jl}(\boldsymbol{x} - \boldsymbol{x}')\varphi_{kl}(\boldsymbol{x}')d\boldsymbol{x}'. \tag{29}$$

Convolution integrals in direct space are simply products in Fourier space. Hence:

$$\hat{\tilde{\varepsilon}}_{ij}(\xi) = \hat{\Gamma}^{sym}_{ijkl}(\xi)\hat{\varphi}_{kl}(\xi), \tag{30}$$

where $\hat{\Gamma}^{sym}_{ijkl} = \mathrm{sym}(\hat{G}_{ik,jl})$. The tensors $\hat{G}_{ik}(\xi)$ and $\hat{\Gamma}^{sym}_{ijkl}(\xi)$ are only functions of $\boldsymbol{L}^0$ and can be readily obtained for every point belonging to $\{\xi^d\}$ (for details, see Lebensohn et al. (2008)). Now, taking into account the definition 27 of $\boldsymbol{\varphi}(\boldsymbol{x})$, Eq. 29 is an integral equation where the velocity gradient appears in

both sides, and, thus, it can be solved iteratively. Assigning initial guess values to the strain-rate field in the regular grid (e.g. $\tilde{\dot{\boldsymbol{\varepsilon}}}\left(\boldsymbol{x}^d\right) = 0 \Rightarrow \dot{\boldsymbol{\varepsilon}}^{(0)}(\boldsymbol{x}^d) = \dot{\bar{\boldsymbol{\varepsilon}}}$), and computing the corresponding stress field $\boldsymbol{\sigma}'^{(0)}(\boldsymbol{x}^d)$ from the local constitutive relation (18) allows to obtain an initial guess for the polarization field in direct space $\boldsymbol{\varphi}^{(0)}(\boldsymbol{x}^d)$ (27), which in turn can be Fourier-transformed to obtain $\hat{\boldsymbol{\varphi}}^{(0)}(\boldsymbol{\xi}^d)$.

The rate of convergence of this fixed point technique is rather poor for nonlinear constitutive relations such as power-law relations between the stress and the strain-rate. Accelerated schemes based on augmented Lagrangians have been proposed to improve this rate of convergence originally by Michel et al. (2000, 2001) for composites, and later adapted by Lebensohn (2001) for polycrystals to which the interested reader in referred for details. Upon convergence, the stress at each material point can be used to calculate the shear rates associated with each slip system (Eq. 4), from which fields of relative activity of the basal, prismatic and pyramidal slip modes can be obtained, as well. While it is certainly possible to use the FFT-based formulation for the prediction of microstructure evolution, in this section we have restricted our analysis to the local fields that are obtained for a fixed configuration. In this sense, the high strain-rate regions predicted by the model (see below) should be regarded as precursors of localization bands. Evidently, microstructural changes that are not considered under this approximation, like the eventual grain's and subgrain's morphologic evolution and rotation, as well as the possible occurrence of local strain hardening, may modify some of the trends observed in the initial micromechanical fields. In order to account for these microstructural changes, the FFT-based formulation has been coupled with the front-tracking numerical platform Elle (Bons et al., 2001). Results of this coupled model are reported in Section 5.2.2.

4.1.2. Application to columnar ice deforming in the secondary creep regime.

Lebensohn et al. (2009) and Montagnat et al. (2011) applied this FFT method to simulate strain rate and stress fields, and local lattice misorientations obtained at secondary creep in columnar ice polycrystals. Lebensohn et al. (2009) compared the simulated fields to a series of compression creep experiments performed by Mansuy et al. (2000, 2002) on laboratory-grown columnar ice samples characterized by multicrystals of controlled shape and orientations. The specimen used for this comparison (see Fig. 13) was a plate of 210×140 mm with a relatively thick (8 mm) section, consisting of a multicrystalline cluster, located in the center of the plate, with c-axes lying on the plane of the plate, and embedded in a matrix of fine-grained ice. This specimen was deformed under a compressive stress of 0.75 MPa exerted vertically in the plane at -10°C under plane strain conditions. Fig. 13 shows, after 0.07 strain, three types of localization bands: basal shear bands, kink bands and sub-boundaries, that change orientation to follow crystallographic directions when they cross from one grain to another.

In this configuration, kink band boundaries are seen mainly inside grains oriented close to 45° from the imposed compression direction. Kink bands, described as a sharp or discontinuous change in orientation of the active slip surface, had been reported in experimental studies conducted on 2-D ice polycrystals (Wilson et al., 1986; Wilson and Zhang, 1994; Montagnat et al., 2011). Sub-boundaries parallel to the c-axis were also observed.

The FFT-based calculation as described in the previous section was run to obtain the overall and local mechanical response of the above-described unit cell representing a columnar ice polycrystal, to the following imposed strain-rate tensor (see also Fig. 14):

$$\dot{\bar{\varepsilon}}_{ij} = \begin{bmatrix} 1\times10^{-8} & 0 & 0 \\ 0 & -1\times10^{-8} & 0 \\ 0 & 0 & 0 \end{bmatrix} \; s^{-1} \qquad (31)$$

The crystallographic texture of the 2-D ice polycrystal consisting of columnar grains with c-axes perpendicular to the axial (vertical) direction x_3 was described in terms of a collection of Euler-angle triplets of the form (φ_1, 90°, φ_2) (Bunge convention). The application of the FFT method required the generation of a periodic unit cell or representative volume Element (RVE), by repetition along x_1 and x_2 of a square domain. This square domain was constructed in such a way that it contained the cross-sections of 200 columnar grains, generated by Voronoi tessellation (see Fig. 14). This square domain is the cross-section of the unit cell, consisting of columnar grains with axes along x_3 and sections in the x_1-x_2 plane. This unit cell was discretized using a 1024×1024×1 grid of regularly-spaced Fourier points, resulting in an average of around 5250 Fourier points per grain. Note that the periodic repetition of this unit cell along x_3 determines infinitely long grains along this direction. Three specific orientations with c-axis respectively at 0°, 45°, and 90° from the compression direction were forced to be among the set of 200 (otherwise random) orientations assigned to the grains. For a plane-strain state, such that x_1 is the tensile direction and x_2 is the compression direction, the grain with φ_1 = 45°(45 deg grain in what follows) is theoretically favorably oriented to deform by soft basal slip, while in the 0 deg and 90 deg grains, the hard pyramidal systems are the only ones favorably oriented to accommodate deformation. It is worth noting that due to the above plane-strain condition and the in-plane orientation of the c-axes, the prismatic slip systems are not well-oriented, for any φ_1 angle.

The computed effective response of this kind of isotropic columnar ice polycrystal deformed in-plane is twice softer compared to an isotropic 3-D polycrystalline ice (Lebensohn et al., 2007). The computed overall relative activities of the different slip modes (i.e. 90.7%, 7.6% and 1.7% for basal, pyramidal and prismatic slip, respectively) show a preeminence of basal slip, a minor contribution of pyramidal slip and a very low activity of prismatic slip. Fig. 15 shows the computed equivalent strain-rate field for the entire unit cell, normalized with respect to the average equivalent strain rate ($\dot{\bar{\varepsilon}}_{eq} = 1.15\times10^{-8}\mathrm{s}^{-1}$). The main feature observed in this plot is a network of high strain-rate bands, precursors of localization bands (in what follows we will

sometimes refer to them simply as "localization bands"). These bands are transmitted from grain to grain and are, in general, inclined with respect to the shortening and extension directions. They follow tortuous paths, sometimes with large deviations from ±45 ° (i.e. the macroscopic directions of maximum shear stress). They follow crystallographic directions (basal poles or basal planes) inside each grain. The predicted bands parallel or perpendicular to the c-axis were reasonably assumed to be kink or shear bands, respectively (see Lebensohn et al. (2009) for details). Some segments of these bands also follow favorably-oriented grain boundaries and frequently go through triple or multiple points between grains, in good agreement with some of the observations of (Mansuy et al., 2002) (Fig. 13). Fig. 16 shows in more details the predicted fields of equivalent strain rate (normalized to $\dot{\bar{\varepsilon}}_{eq}$), equivalent stress (in units of τ^{bas}) and relative basal activity, in the vicinities of the 45 deg grain. Two very intense (i.e. local strain rates higher than 10 times the macroscopic strain rate) and parallel kink bands are seen inside the 45 deg grain, connected by several less intense shear bands (orthogonal to the pair of kink bands, lying on to the basal plane), in good agreement with Mansuy's experiments (see Fig. 13). The basal activity in the 45 deg grain is very high, although some regions of high non-basal activity can be observed between shear bands and immediately outside the kink bands. The latter is compatible with a low or even vanishing resolved shear stress on basal planes in those locations, which may be responsible for the formation of basal dislocation walls that are at the origin of a kink band (Mansuy et al., 2002). This correlation between kink band precursors and nearby localized high non-basal activity is systematic in these results. From the same detailed analysis performed around the 0 and 90 deg grains, a good match was found with experimental observations.

In (Montagnat et al., 2011), the viscoplastic FFT-based approach was applied to the exact experimental microstructure of a compressive test performed on a 2D columnar sample. Samples (dimensions $\approx 10{\times}10{\times}1.5$ cm^3) were grown in the laboratory under a uniaxial temperature gradient to reach a columnar microstructure with all c-axes lying parallel to the sample surface. In this work, the observed kink bands could be associated with misorientations between adjacent regions of a grain interior of more than 5°, and their exact nature in term of dislocation arrangements were confirmed by EBSD measurements. Although the boundary conditions of the modeling were slightly different from the experimental one, the model was able to predict the exact location of the localization bands. The bands were associated with stress concentration that could reach five times the applied macroscopic stress, and to high levels of local non basal activity (see Fig. 17). Nevertheless, the amplitude of the modeled lattice misorientations were always overestimated, and this was associated with the fact that very local grain boundary migration and new grain nucleation (dynamic recrystallization mechanisms) observed experimentally were not considered in the model (see Section 5.2.2).

4.2. Elasto-viscoplastic FFT approach

The full-field FFT approach described above has been extended very recently to the case of elasto-viscoplasticity, see (Idiart et al., 2006; Suquet et al., 2011; Lebensohn et al., 2012). As for purely viscoplastic behaviors, its application to highly anisotropic material like ice allows investigating the accuracy of elasto-viscoplastic mean-field models (see Section 3.6) since, as already mentioned, the FFT technique provides the "exact" (in a numerical sense) response of the specimen with the actual microstructure and local constitutive relations. Application to ice allows studying transient creep effects with more detail. Comparison with experimental strain field measured with an intragranular spatial resolution has been provided in (Grennerat et al., 2012) making use of the relative ease of producing samples with controlled 2-D microstructure, compared to other polycrystalline materials.

4.2.1. The mechanical problem

The method described in Suquet et al. (2011) considers the same microstructure description as in Section 4.1: a polycrystalline volume V composed of several grains of different orientations, each grain obeying constitutive relations defined in Section 2.3. The volume V is subjected to a macroscopic loading path, which can be a prescribed history of average strain, or a history of average stress or a combination of both. For simplicity, the method is presented here assuming a prescribed history of macroscopic strain $\overline{\boldsymbol{\varepsilon}}(t), t \in [0,T]$. Other types of loadings can be handled by different methods described in (Michel et al., 1999) for instance.

The local problem to be solved to determine the local stress and strain fields in the volume element V consists of the equilibrium equations, compatibility conditions, constitutive relations and periodicity boundary conditions:

$$\left\{\begin{array}{ll} (\dot{\boldsymbol{\sigma}}, \dot{\boldsymbol{\tau}}_0, \dot{\boldsymbol{X}}) = \boldsymbol{F}(\dot{\boldsymbol{\varepsilon}}, \boldsymbol{\sigma}, \boldsymbol{\tau}_0, \boldsymbol{X}, \boldsymbol{x}, t), & \text{for } (\boldsymbol{x},t) \in V \times [0,T], \\ \boldsymbol{\varepsilon}(\boldsymbol{x},t) = \frac{1}{2}(\nabla \boldsymbol{u}(\boldsymbol{x},t) + {}^{\mathsf{T}}\nabla \boldsymbol{u}(\boldsymbol{x},t)), & \\ \text{div}\, \boldsymbol{\sigma}(\boldsymbol{x},t) = 0 & \text{for } (\boldsymbol{x},t) \in V \times [0,T], \\ \boldsymbol{u}(\boldsymbol{x},t) - \overline{\boldsymbol{\varepsilon}}(t).\boldsymbol{x} \text{ periodic on } \partial V, & \text{for } t \in [0,T] \end{array}\right. \tag{32}$$

The data of interest are the effective response $\overline{\boldsymbol{\sigma}}(t),\ t \in [0,T]$ of the polycrystal, the history of the average strain $\overline{\boldsymbol{\varepsilon}}(t),\ t \in [0,T]$, but also the local fields $\boldsymbol{\sigma}(\boldsymbol{x},t), \boldsymbol{\varepsilon}(\boldsymbol{x},t)$ and other significant fields (internal variables, thermodynamic forces etc....).

The extension of the simplest version of the FFT-based method, also called the *basic scheme*, to constitutive relations including crystalline elasto-viscoplasticity relies on two ingredients:

1. A time-integration scheme for the constitutive differential equations. The time interval of interest $[0,T]$ is split into time steps $[t_n, t_{n+1}]$. All quantities are assumed to be known at time t_n, and the quantities at time t_{n+1} are unknown. This time integration is performed at every point $\mathbf{x}^d$ of the discretized polycrystal and the evolution problem is reduced to a problem for the stress and strain fields $\boldsymbol{\sigma}$ and $\boldsymbol{\varepsilon}$ at time t_{n+1} in the form
$$\boldsymbol{\sigma}_{n+1}(\mathbf{x}^d) = \boldsymbol{\mathcal{F}}_{\boldsymbol{n+1}}(\mathbf{x}^d, \boldsymbol{\varepsilon}_{n+1}(\mathbf{x}^d)) \tag{33}$$
2. A FFT global scheme to solve the local problem for a nonlinear composite obeying Eq. (33).

The algorithm developed applies to a wide class of constitutive relations, see (Suquet et al., 2011). As before, it is limited to specimens submitted to periodic boundary conditions. Results presented below are performed with the FFT-based program Craft (freely available at http://craft.lma.cnrs-mrs.fr). For application to elastoviscoplasticity in ice, the local constitutive relation is the one provided above, see Eqs (2, 3, 6, 7, 8). It can also be formulated via the following differential equation:

$$\dot{\boldsymbol{Y}} = \boldsymbol{F}(\dot{\boldsymbol{\varepsilon}}, \boldsymbol{Y}, t), \tag{34}$$

where

$$\boldsymbol{Y} = \begin{pmatrix} \boldsymbol{\sigma} \\ \tau_0^{(k)},\ k = 1, ...M \\ X^{(k)},\ k = 1, ...M \end{pmatrix},$$

$$\boldsymbol{F}(\dot{\boldsymbol{\varepsilon}}, \boldsymbol{Y}, t) =$$

$$\begin{pmatrix} \boldsymbol{C} : \left(\dot{\boldsymbol{\varepsilon}} - \sum_{k=1}^{M} \dot{\gamma}^{(k)}(\boldsymbol{Y}) \boldsymbol{\mu}^{(k)} \right) \\ \left(\tau_{sta}^{(k)} - \tau_0^{(k)} \right) \sum_{\ell=1}^{M} h^{(k,\ell)} \left| \dot{\gamma}^{(\ell)}(\boldsymbol{Y}) \right| \\ c^{(k)} \dot{\gamma}^{(k)}(\boldsymbol{Y}) - d^{(k)} X^{(k)} \left| \dot{\gamma}^{(k)}(\boldsymbol{Y}) \right| - e^{(k)} \left| X^{(k)} \right|^m \mathrm{sign}\left(X^{(k)} \right) \end{pmatrix} \tag{35}$$

with $\boldsymbol{C}$ the elastic stiffness ($\boldsymbol{C} = \boldsymbol{S}^{-1}$). The set of parameters used are given in Table 1.

4.2.2. Application to strain field prediction in a 2D-1/2 configuration.

The elasto-viscoplastic FFT approach was used to predict strain and stress field evolution during transient creep tests on ice polycrystals, in comparison with experimental measurements performed by Grennerat et al. (2012).
Samples were grown following (Montagnat et al., 2011) (see Section 4.1.2). This way, when compressed, (i) plastic deformation can be approximate as 2-D, and (ii) strain fields measured at the specimen surface are representative for the sample volume owing to the minimisation of in-depth microstructure gradients. Average grain size (section perpendicular to the column direction) was about 5 mm and most of the c-axes were oriented parallel to the surface (± 15°). The microstructure and grain orientation were measured using an Automatic Fabric Analyzer (Russell-Head and Wilson, 2001) which provides orientation values with about 50 μm resolution, and 1° accuracy. A Digital Image Correlation technique (Vacher et al., 1999) was applied to measure the strain heterogeneities on the surface perpendicular to the column direction. From displacement measurements performed during transient creep in ice, i.e. up to 1 to 2%, at -10°C, under 0.5 MPa, strain fields were evaluated with a resolution of about 0.2%, and at a spatial resolution of about 1 mm.

The experimental microstructures were implemented in the code using the fabric analyzer data of 2000×2000 pixels (but the model input does not need to be square). One pixel in the third dimension (column direction) is enough to reproduce the 2D-1/2 geometry thanks to the periodic boundary conditions.

Fig. 18 presents the strain field measured experimentally at the end of the transient creep, and the simulated fields of strain and stress. Although simulated boundary conditions did not precisely match the experimental ones, the heterogeneities of the strain field that develop during transient creep of polycrystalline ice were reproduced well by the model (Grennerat et al., 2012). In particular, the model was able to reproduce the characteristic length of the heterogeneities being larger than the grain size, and scaling with the sample dimensions. Furthermore, both experimental and modeled results showed no correlation between the orientation of the c-axis and the strain intensity (see Fig. 19). This result casts doubt on the relevance of the distinction between "hard grains" and "soft grains" classically made for the analysis of ice mechanical behavior, and more generally for anisotropic materials.
Fig. 20 represents the evolution of the simulated equivalent strain field from 0.25 to 0.60% of compression during transient creep. As observed experimentally, the strain heterogeneities develop early during the transient creep and are reinforced up to about 10 times the imposed strain.

5. Modeling of dynamic recrystallization mechanisms

Under laboratory conditions (described in Section 1), dynamic recrystallization (DRX) dominates the changes of microstructures and fabrics in the tertiary creep regime, that is after about 1% macroscopic strain (Duval, 1981; Jacka and Maccagnan, 1984; Jacka and Li, 1994). During DRX, grain nucleation and grain boundary migration are two processes that contribute to the reduction of the dislocation density, therefore of the stored deformation energy (Humphreys and Hatherly, 2004). In the laboratory, tertiary creep is a continuous sequence of deformation and recrystallization that gradually results in a steady state. This steady state is associated with an equilibrium grain size (Jacka and Li, 1994) and a girdle-type fabric with c-axes at about 30° from the compression axis (Jacka and Maccagnan, 1984), or with two maxima in simple shear (Bouchez and Duval, 1982).
In polar ice sheets, DRX was identified from observation on ice thin sections along ice cores (Alley, 1992; Thorsteinsson et al., 1997; de la Chapelle et al., 1998; Kipfstuhl et al., 2006). Three regions are usually defined: (i) normal grain growth driven by the reduction of grain-boundary energy in the upper hundreds meters of the core, (ii) rotation recrystallization during which new grains are formed by the progressive lattice rotation of the subgrains in the main part of the core and (iii) migration recrystallization similar to the one observed in the laboratory, in the bottom part where the temperature is above -10°C (see Montagnat et al. (2009) and Faria et al. (this issue) for a review).

Recrystallization and grain growth significantly influence the microstructure, the fabric and therefore the mechanical properties. To be able to integrate these mechanisms in the modeling

of ice deformation is therefore crucial for an accurate prediction of its behavior.

5.1. Dynamic recrystallization within mean-field approaches

Several attempts were made to integrate dynamic recrystallization mechanisms into mean-field approaches as described in Section 3.

On the basis of the VPSC scheme (tangent version) described in Section 3 for the description of the mechanical behavior, Wenk et al. (1997) developed a nucleation and grain-growth model to represent DRX in anisotropic materials such as ice. The model is based on the hypothesis that grains with a high stored energy (highly deformed) are likely to nucleate new grains and become dislocation-free. They may also be invaded by their neighbors which have a lower stored energy. Depending on the respective importance of nucleation and grain boundary migration processes, the recrystallization textures are expected to favor either highly deformed components or less deformed ones.

One must first remember that, in the VPSC scheme, grains are represented by inclusions in an homogenous equivalent medium (HEM). Grain interactions are therefore represented "averaged" through the interaction between the inclusion and the HEM.

In this model, nucleation is represented, by a probability of nucleation P per time increment Δt for each deformation step:

$$P \propto \Delta t \times exp(-A/E^2) \tag{36}$$

The constant A depends on the grain boundary energy and was taken as an adjustable parameter. E is a proxy of the stored energy, $E \propto \sum_s (\Delta\tau_0^s)$ with $\Delta\tau_0^s$ the variation of the critical resolved shear stress on the system s during the deformation step. This calculation supposes a hardening law for each slip system to be defined. An isotropic hardening law was chosen in the form $\dot{\tau}_0^s = H \sum_s \dot{\gamma}^s$, with hardening matrix H being isotropic. A threshold was then defined for the minimum strain energy to nucleate, and the new grain completely replaced the old one (same size, same orientation), with a stored energy equal to zero.

The grain boundary migration rate was taken proportional to the difference in stored energy between the grain and the average, i.e., the HEM. The development of the microstructure is therefore a balance between nucleation and growth. Adjustable parameters were varied arbitrarily for comparison purpose.

Applied to ice, this model resulted into weaker fabrics than the one obtained by the classical VPSC tangent approach, mostly because grains near the compression axis disappeared (high stored energy) and only a few girdle grains, and a few grains exactly aligned with the compression axis from the beginning, remained.

Thorsteinsson (2002) included some DRX in its Nearest-Neighbor Interaction (NNI) model described in Section 3. Polygonization associated with rotation (or continuous) recrystallization is accounted for by comparing the resolved shear stress in the crystal ($|\sum_s \tau_s \hat{\boldsymbol{b}}_s|$) to the applied stress (with τ_s the shear stress on system s, and $\hat{\boldsymbol{b}}_s$ a unit vector in the direction of the Burgers vector). If the ratio is smaller than a given value, and the dislocation density higher than a given value, then the crystal size is halved and both new grains are rotated by a fixed $\Delta\theta$ of 5°. Grain growth occurs by normal grain growth according to (Gow, 1969; Alley et al., 1986) parabolic law ($D^2 - D_0^2 = Kt$). The grain growth factor K follows an Arhenius-type dependence on the temperature. To take into account the grain growth associated with the difference in dislocation-stored energy between the grain i and the average, this growth factor was modified into ($\tilde{K} = (E_{disl}^{av} - E_{disl}^{i})K'$) with K' a constant depending on temperature and impurities.

Migration recrystallization is included in the model by considering the balance between grain-boundary energy, and stored energy associated with dislocations (the stored energy is calculated following (Wenk et al., 1997), as just described, and translated into dislocation density). A crystal recrystallizes (i.e. is replaced by a crystal with initial dislocation density ρ_0) when the dislocation energy is higher than the grain-boundary energy. This assumption relies on the hypothesis that stored energy is released by normal grain growth (driven by GB energy), and that dynamic recrystallization only occurs if this relaxation in not efficient enough to decrease the dislocation density. The size of the new crystal is adjusted with the effective stress following (Guillopé and Poirier, 1979; Jacka and Li, 1994) and its orientation is chosen at random in the range of the "softest" orientations in the applied stress state.

Modeling results were obtained for comparison to a case similar to the GRIP ice core, with vertical compression, and rotation recrystallization dominating. The introduction of polygonization allows for the preferential removal of "hard" grains, which leads to a weaker fabric compared to the "no-recrystallization" case. In particular, when associated with the NNI formulation, the model is able to reproduce fabrics quite similar to those measured along the GRIP ice core at several depths. "Girdle-type" fabric similar to the experimental fabrics, results from the introduction of migration recrystallization. However, parametrization remains weak, in particular the estimation of the dislocation density, and of the recrystallized grain orientations.

The last example presented here is the cellular automaton model for fabric development by Ktitarev et al. (2002) and Faria et al. (2002). The application was mostly to reproduce the fabric measured along deep ice cores, with the assumption of deformation under uniaxial compression. The cellular automaton (CA) frame is especially suitable for simulation of systems represented by a certain number of cells, which are associated with generalized state variables and arranged in regular environment. The considered material is a thin horizontal layer of ice located along the ice core, thin enough so that it is considered homogeneous in the vertical dimension. To discretize the problem according to the CA method, the authors took a one-dimensional lattice of equal cells representing the grains, described by their size, and their orientation. The basic dynamical quantity of the algorithm is the dislocation density. This density increases with deformation and depends

on the orientation of the grain. Recrystallization mechanisms proceed when a critical value is reached. Normal grain growth is accounted for following Gow (1969) and is apparently the only growth mechanism associated with polygonization mechanisms. The increase in dislocation density is associated with the resolved shear stress on the basal system and the recrystallization model developed in Montagnat and Duval (2000) is used to estimate the evolution of the density in relation with grain size and polygonization mechanisms. Rotation of grains is ruled by a kinematic equation based on the inelastic spin, assuming a compressive stress proportional to the depth along the core, and a linear dependence between the shearing rate of sliding on the basal system and the resolved shear stress. The time evolution was related to the depth along the core using the Dansgaard et al. (1993) relation. Following Duval and Castelnau (1995), migration recrystallization was only applied bellow 2800 m depth. During migration recrystallization, new grains were allowed to grow much faster by consuming up to ten cells at every time step, until it is impinged by another growing grain, or until it reaches the critical size of the steady state.

The model was able to provide a good qualitative evolution of the grain size, by separating the influence of normal grain growth, polygonization and migration recrystallization similarly to what was suggested from the measurements along the GRIP ice core (Thorsteinsson et al., 1997; de la Chapelle et al., 1998). Concerning the fabric evolution, the model was able to predict the evolution toward a single maximum, but the kinetics is too strongly influenced by the polygonization, and further by migration recrystallization.

5.2. Dynamic recrystallization within full-field approaches

This section presents a coupling between a platform for structural change in materials (Elle) with the full-field FFT approach presented in Section 4.1, to predict the microstructure evolution of ice polycrystals during dynamic recrystallization. A critical step in the development of generic models linking plastic deformation and recrystallization is the incorporation of the interaction between intra- and intergranular heterogeneities of the micromechanical fields (i.e. strain rate and stress) and the recrystallization processes. Because local rotations of the crystal lattice are controlled by local gradients of plastic deformation, heterogeneous distributions of lattice orientations are observed at the grain and subgrain scale, see Section 4. This has a strong influence on recrystallization as this is a process driven by the local gradients of energy (e.g. grain boundary or stored strain energy). Traditional mean-field models used to predict microstructure evolution during recrystallization are based on a simplified description of the medium and cannot fully describe intragranular heterogeneities (Section 5.1). Therefore, explicit full-field approaches are required for a better understanding of dynamic recrystallization and prediction of microstructure evolution at large strain.

5.2.1. The Elle modeling plateform

Elle is a platform for the numerical simulation of processes in rocks and grain aggregates, with particular focus on (micro-) structural changes (Jessell et al., 2001; Jessell and Bons, 2002; Bons et al., 2008; Piazolo et al., 2010). The simulations act on an actual 2D image of the microstructure (Fig. 21.a). Elle is currently restricted to 2D cases although the underlying principles for 2D are equally valid in 3D (Becker et al., 2008), and therefore the approach could be converted for 3D simulations.

The central philosophy of Elle is to enable the coupling of processes that act on the material, recognizing that the effect of one process may significantly alter that of a concurrent process. Dynamic recrystallization, for example, can greatly change crystallographic preferred orientations in mineral aggregates deforming by dislocation creep (Jessell, 1988a,b). Coupling of processes is achieved in Elle using the principle of operator splitting, whereby individual processes successively act on the model in isolation, for a small time step. This approach greatly simplifies coding, as the coupling between processes needs not be programmed itself, but emerges from their alternating effect on the model.

Each process in Elle is an individual program or module. A shell-script takes the starting model and then passes it in a loop to the individual processes, which each in turn modify the model slightly. Each loop represents one time step. The user can freely determine the mix of processes that operate by choosing which ones to include in the loop. The relative activity of individual processes is determined by the parameters passed on to each process.

The model is essentially defined by two types of nodes: boundary nodes (bnodes) and unconnected nodes (unodes) (Fig. 21.b). Bnodes define the boundaries of a contiguous set of polygons (termed flynns). These flynns typically represent single grains, but can also represent regions within a material, for example rock layers (Llorens et al., 2012). The boundaries of the flynns are formed by straight segments that connect neighboring bnodes. One bnode can be connected by either two or three other bnodes. The use of bnodes and flynns makes the model suitable for a range of Finite Element and front-tracking models.

Unodes form a second layer of the model. These are nodes that do not necessarily have fixed neighborhood relationships and typically represent points within the material. Some processes are not amenable to be modeled with polygons, but are best simulated with a regular grid of unodes. The FFT code is an example. Nodes and flynns can have a range of attributes assigned to them, such as c-axis orientation, boundary properties, etc.

Elle uses fully wrapping boundaries. A flynn that touches one side of the model continues on the other side (Fig. 21.a). The model is thus effectively a unit cell that is repeated infinitely in all directions. Although Elle typically uses a square model, deformation may change the unit cell into a parallelogram shape.

Elle now includes a large and ever growing number of process modules for a variety of processes that mostly relate to mi-

crostructural developments in mineral aggregates. Each process can essentially act on the model in only two ways: changing the position of a node (e.g. a bnode in case of grain boundary migration) or changing the value(s) of attributes of flynns or nodes (e.g. concentration at a unode in a diffusion simulation). Some of the most relevant current processes are:

- Normal grain growth driven by the reduction of surface energy, and hence curvature of grain boundaries. This process was used by Roessiger et al. (2011) to address the issue of the competition between grain growth and grain size reduction in the upper levels of polar ice caps (Mathiesen et al., 2004). Surface energy can be anisotropic, i.e. depending on the lattice orientation of the grains on either side of the boundary (Bons et al., 2001). Two-phase grain growth has been applied to grain growth in rocks with a small proportion of melt (Becker et al., 2008) and to ice with air bubbles (Fig. 21)(Roessiger et al. this volume).

- The Finite Element module Basil is used for incompressible power-law viscous deformation (Barr and Houseman, 1996). Using viscosities that are assigned to flynns, it calculates the stress and velocity fields resulting from applied boundary conditions. It has been used to study the behaviour of rigid inclusions in a deforming matrix (Bons et al., 1997), the behaviour of deforming two-phase materials as a function of viscosity contrast and composition (Jessell et al., 2009) and for folding of layers (Llorens et al., 2012). The wrapping boundaries of the Elle model, in combination with continuous remeshing allows for arbitrarily high strains (Jessell et al., 2009). In combination with grain growth and dynamic viscosity, Jessell et al. (2005) studied strain localisation behaviour. Durand et al. (2004) investigated the influence of uniaxial deformation on grain size evolution in polar ice cores and its influence on ice dating methods.

- Dynamic recrystallisation includes grain boundary migration driven by strain energy (dislocation density) and the formation of new grain boundaries by progressive subgrain rotation or polygonisation (Urai et al., 1986). In the next section (5.2.2) we will describe how these processes, employing a front-tracking model for grain-boundary migration, are linked with the FFT approach (Griera et al., 2011, 2012; Piazolo et al., 2012) to model the stress and strain-rate fields and the driving forces for recrystallization.

- A final Elle module of potential relevance to ice is that developed by Schmatz (2010) for the interaction between migrating grain boundaries and small particles (e.g. dust or clathrates). The particles are represented by unodes, which, when swept by a grain boundary, can latch onto that boundary. Particles can slow down grain boundary movement, but can also be dragged along and eventually be released by a grain boundary.

Summarizing, Elle provides a large number of routines to simulate grain-scale processes in minerals and rocks, and hence in glacial or polar ice. The open and versatile code allows for more process modules to be added or existing ones to be tailored for application to ice. A significant advantage of the code is that it enables the investigation of the complex microstructural and mechanical effects of multiple, concurrent and coupled processes.

5.2.2. Coupling Elle platform to FFT approach

Most of the numerical approaches used to simulate deformation and microstructural evolution of rocks and metals are based on combining deformation approaches based on the Finite Element Method with Monte Carlo, cellular automaton, phase field, network or level-set methods to simulate recrystallization (Jessell, 1988a,b; Raabe and Becker, 2000; Piazolo et al., 2002, 2010, 2012; Solas et al., 2004; Battaile et al., 2007; Logé et al., 2008). An alternative to these methods is the numerical scheme used in this study based on the coupling between the crystal plasticity FFT-based code (Lebensohn, 2001) (section 4.1) and the Elle modeling platform just described (Bons et al., 2008). Both codes have been previously explained and here we only concentrate on some particularities of the coupling between them. The FFT-based formulation is integrated within the Elle plateform using a direct one-to-one mapping between data structures. The polycrystalline aggregate is discretized into a periodic, regular array of spaced and unconnected nodes (Fourier Points in the FFT and "unconnected nodes", unodes, in Elle; Fig. 21).

Numerical simulation is achieved by iterative application of small time steps of each process. After numerical convergence of the FFT model, data is transferred to Elle assuming that the micromechanical fields are constant in the incremental time step. The position and material information of unodes are directly updated because they are equivalent to the Fourier points, while position of boundary nodes (bnodes) are calculated using the velocity field. Based on the evolution of the predicted local lattice rotation field, the dislocation density can be estimated using strain gradient plasticity theory (e.g. Gao et al. (1999); Brinckmann et al. (2006)) or using the dislocation density tensor or Nyes tensor (Nye, 1953; Arsenlis and Parks, 1999; Pantleon, 2008). With this approach, only geometrical necessary dislocations required to ensure strain compatibility are estimated. To simplify the problem, we use a scalar approach where all dislocations are assumed to be related to the basal plane. The lattice-orientation and dislocation-density fields provide the input parameters to predict recrystallization in the aggregate.

Recrystallization is simulated by means of three main processes: nucleation, subgrain rotation and grain boundary migration. Using the kinematic and thermodynamic instability criteria of classical recrystallization theory (Humphreys and Hatherly, 2004; Raabe and Becker, 2000), nucleation is simulated by the creation of a small new, dislocation-free flynn when the local misorientation or dislocation density exceeds a defined threshold. The lattice orientation of the new grain is set to that of the critical unode. When a cluster of unodes within a grain

share the same orientation that is different from the rest of the unodes in that grain, a new grain boundary is created, while preserving the lattice orientations of the unodes. A technical limitation is that nucleation of grains and subgrains is only allowed along grain boundaries. Nucleation within grains are therefore not possible.

Grain boundary migration is described by a linear relationship between velocity (v) and driving force per unit area (Δf), by $v = M\Delta f$ where M corresponds to the grain boundary mobility, which has an Arhenius-type dependency on temperature. Grain boundary curvature and stored strain energies are used as driving forces for grain boundary motion. For this situation, the driving force can be defined as

$$\Delta f = \Delta E - 2\gamma/r \tag{37}$$

where ΔE is the difference of stored strain energy across the boundary, γ is the boundary energy and r is the local radius of curvature of the grain boundary. Stored strain energy is the energy per unit volume associated with lattice distortions and depends on the dislocation density (ρ) and dislocation type. Grain boundary motion is simulated using the free-energy minimization front-tracking scheme of (Becker et al., 2008). When an unode is swept by a moving grain boundary, it is assumed that dislocations are removed and the new lattice orientation is that of the nearest unode belonging to the growing grain.
Following the Elle philosophy, each process runs individually, following a pre-established sequence. After all Elle processes have run, the unodes layer is used to define the new input microstructure to be deformed viscoplastically by the FFT code. A drawback is that the unodes are not following a regular mesh, a requirement needed by the FFT approach. For this reason, as proposed by Lahellec et al. (2003), and later adapted in the context of Elle by Griera et al. (2011, 2012), a particle-in-cell method is used to remap all material and morphological information to a new regular computational mesh. In order to avoid unrealistic crystallographic orientations, these are not interpolated during remapping. The crystallographic orientation of a new Fourier Point that belongs to a specific grain is that of the nearest unode that belongs to the same grain. This allows to run numerical simulation up to large strains.

5.2.3. Application to creep experiments and natural ice

An example of numerical simulation using the FFT/Elle approach is shown in Fig. 22. The simulation is based on a creep experiment of polycrystalline columnar ice. Samples and experimental conditions are those of (Montagnat et al., 2011) described in Section 4.1.2. The specimen was deformed at -10°C under uniaxial conditions with a constant load of 0.5 MPa up to an axial strain of 4%. A thin section of the initial and the final microstructure was analyzed using the Automatic Ice Texture Analyzer method (Russell-Head and Wilson, 2001) to obtain the local c-axis orientations. After a 4% of shortening, the onset of local recrystallization is evident in the experiment (Fig. 22a), in the form of irregular and serrated grain boundaries and small new grains that are preferentially located at triple junctions and along grain boundaries. Localized variations in the orientation of the basal plane form sharp and straight subgrain boundaries that indicate intracrystalline deformation. The experimental c-axis map was used as input for the FFT/Elle simulation. The experimental starting microstructure was discretized into a grid of 256×256 Fourier points. As only the c-axis orientation is known, the other axes are given a random orientation. Crystal plasticity is described with an incompressible rate-dependent equation for basal, prismatic and pyramidal slip (see Section 3). Critical resolved shear stress for basal slip was set 20 times lower than for non-basal systems. The physical properties used for recrystallization are as follows: mobility $M = 1 \times 10^{-10}$ m^{2}Kg^{-1}s^{-1} (e.g. Nasello et al., 2005), isotropic boundary energy $\gamma = 0.065$ Jm^{-2} (Ketcham and Hobbs, 1969), shear modulus $G = 3{\times}10^9$ Pa and critical dislocation density $\rho = 1 \times 10^{12}$ m^{-2} (de la Chapelle et al., 1998). Pure shear boundary conditions were imposed with vertical constant strain rate of -1×10^{-8} s^{-1} up to a 4% of strain in 1% increments.

The computed orientation map and grain boundary misorientation are shown in Fig. 22b. Several features of the experiment are seen in the numerical simulation, such as the development of sharp misorientations or kink bands, bulging and serrated grain boundaries, and new grains at triple junction and grain boundaries. There is a good correlation between location of kink bands in the experiment and the simulation. However, the width of kink bands in the simulation is dependent on the numerical resolution. A relationship between grain boundary motion/nucleation and high dislocation-density regions is observed (Fig. 23). Variations in dislocation densities across grain boundaries lead to migration of these boundaries in the direction of the dislocation density gradient. However, some discrepancies are also seen, such as, for example, grain boundary motion (e.g. at the bottom-left part) that was not observed in the experiment. One explanation may be that low and high angle grain boundaries were not differentiated in the simulation and, therefore, both types had similar mobility.

A second example aims to show the strong effect recrystallization can have on the final microstructure. A 10×10 cm^2 microstructure with 1600 grains with random c-axis orientations (Fig. 24a) was deformed to 40% shortening in plane-strain pure shear. The values of mechanical (slip systems, CRSS, etc) and recrystallization (mobility, surface energy, etc) properties are similar to those of the model described before, but adjusted to a natural strain rate of 10^{-12} s^{-1} at about -30°C. Fig. 24b shows the c-axis and relative misorientation maps for an extreme case with no recrystallization (FFT only). Dominant red and purple colors indicate that the c-axis of crystallites are preferentially oriented at low angles to the shortening direction. Elongated grains are oriented parallel to the stretching direction. Remarkable differences are observed when recrystallization is activated (Fig. 24c). Grain boundaries are smooth and grains larger and more equidimensional. Despite the significant difference in microstructure, both simulations show a single maximum c-axis distribution at low angle to the shortening direction. The strong resemblance of the simulated microstructure with that of natural ice (Thorsteinsson et al., 1997; de la Chapelle et al., 1998; Weikusat et al., 2009) shows the strong potential of modeling of

ice deformation based on an actual map of the microstructure.

6. Toward large scale ice flow modeling

A number of models have been developed in glaciology to simulate the flow of anisotropic ice and the strain-induced development of fabric within polar ice-sheets. Accounting for ice anisotropy in an ice-flow model implies to (i) build a macroscopic anisotropic flow law whose response will depend on the local fabric and (ii) have a proper description of the ice fabric at each node of the mesh domain and be able to model the fabric evolution as a function of the flow conditions. We hereafter present the main issues to address these two points.

Due to the scale of these large ice-masses, the implementation of a polycrystalline law must stay simple enough and numerically tractable. At present, full-field or even homogenization models presented previously are computationally too demanding and cannot realistically be used to estimate the mechanical response in an ice-sheet flow model. Here we present two approaches to build a simple and efficient macroscopic law for polycrystalline ice. The first one is based on the concept of a scalar enhancement factor function so that the collinearity between the strain-rate and the deviatoric stress tensors is conserved (Placidi and Hutter, 2006), see Section 6.1. The second polycrystalline law is fully orthotropic and depends on six relative viscosities, function of the fabric (Gillet-Chaulet et al., 2005, 2006), see Section 6.2. Both models are phenomenological and must be calibrated using experimental or numerical results, as described below.

With regards to other materials, the advantage of the hexagonal symmetry of ice is that the crystal rheology can be assumed transversally isotropic (only true for a linear rheology). Under this assumption, only one unit vector suffices to describe the lattice orientation, thus simplifying the mathematical description of fabrics. With regards to other materials, the advantage of the hexagonal symmetry of ice is that only one unit vector suffices to describe the lattice orientation, thus simplifying the mathematical description of fabrics. The discrete description of the fabric, *i.e.* a couple of angles for each crystal, would require too large a number of variables to be stored at each node of the domain mesh. Typical mesh size are hundreds of thousand nodes in 3D (Seddik et al., 2011) up to few millions for the most recent applications (Gillet-Chaulet et al., 2012). The use of a parameterized orientation distribution function (ODF) would decrease the number of parameters, but evolution equations for these parameters to describe the fabric evolution cannot be obtained in a general case (Gagliardini et al., 2009). The orientation tensors, which describe the fabric at the macroscopic scale in a condensed way are more suitable. Five parameters are needed to describe an orthotropic fabric (the two eigenvalues of the second-order orientation tensor and the three Euler angles to specify the position of the material symmetry basis), and an evolution equation for the second-order orientation tensor can be easily derived from the macroscopic stress and strain-rate fields.

6.1. Continuous Diversity and the CAFFE model

The CAFFE model (Continuum-mechanical Anisotropic Flow model based on an anisotropic Flow Enhancement factor) results from a suitable combination of two basic concepts: a power law description of ice rheology resembling the well-known Glen's flow law (Glen, 1955); and a multiscale approach to model the evolution of the polycrystalline microstructure of ice based on the general theory of continuous diversity (Faria, 2001; Faria and Hutter, 2002; Faria et al., 2003).

The ideas leading to the CAFFE model have been elaborated in a series of works by Luca Placidi and his collaborators (Placidi, 2004, 2005; Placidi and Hutter, 2005, 2006; Placidi et al., 2004). These ideas culminated in the definitive CAFFE formulation, presented by Placidi et al. (2010), in which the so-called enhancement factor of Glen's flow law becomes a function of the material anisotropy (fabric), and the evolution of the latter is governed by an orientation-dependent mass balance equation derived from the theory of continuous diversity applied to glacier and ice-sheet dynamics (Faria, 2006a,b; Faria et al., 2006).

The greatest strength of the CAFFE model is its successful compromise between accuracy and flexibility, which allows one to upgrade existing computer models of isotropic ice-sheet dynamics based on Glen's flow law into efficient anisotropic models, without profound changes in the original code. In fact, due to its relative simplicity, the CAFFE model has already been implemented in several numerical ice-flow simulations. For instance, it has been used by Seddik et al. (2008) and Bargmann et al. (2011) to simulate the ice flow at the site of the EPICA-DML drill site at Kohnen Station, Dronning Maud Land, East Antarctica, while Seddik et al. (2011) used it to simulate the ice flow in the vicinity of the Dome Fuji drill site in central East Antarctica.

In the following, we review the CAFFE formulation presented by Placidi et al. (2010). The fundamental idea is to regard polycrystalline ice as a "mixture" of lattice orientations, following the philosophy of the theory of Mixtures with Continuous Diversity (MCD) proposed by Faria (2001, 2006a). Succinctly, a mixture with continuous diversity is a multicomponent medium made up of an infinite number of mutually interacting species, whose distinctive properties vary smoothly from one to another.

In the case of polycrystalline ice, species are distinguished by their c-axis orientations. Each point of the continuous body is interpreted as a representative volume element, which encompasses a large number of crystallites with their own c-axis orientations. Each of such orientations is mathematically identified with a point on the surface of the unit sphere $\mathcal{S}^2$ and represented by a unit vector $\boldsymbol{n} \in \mathcal{S}^2$. As a consequence, for each species one can introduce a mass density field $\varrho^*(\boldsymbol{x}, t, \boldsymbol{n})$, given at a certain position $\boldsymbol{x}$ within the polycrystal, and at time t, sometimes called orientational mass density, such that, when integrated over the whole unit sphere, the usual mass density field of the polycristal (i.e. of the "mixture") results:

$$\varrho(\boldsymbol{x}, t) = \int_{\mathcal{S}^2} \varrho^*(\boldsymbol{x}, t, \boldsymbol{n}) \, \mathrm{d}^2 n \, , \tag{38}$$

where d^2n (= $\sin\theta \mathrm{d}\theta\mathrm{d}\phi$ in spherical coordinates) is the infinitesimal solid angle on the unit sphere $\mathcal{S}^2$. The product $\varrho^*(\boldsymbol{x}, t, \boldsymbol{n})\,\mathrm{d}^2n$ is the mass fraction of crystalline material in the volume element with c-axis directed towards $\boldsymbol{n}$ within the solid angle d^2n. Therefore, assuming that the material is incompressible, the mass (or volume) fraction ϱ^*/ϱ can be interpreted as the usual orientation distribution function (ODF) in the context of materials science (Bunge, 1993; Zhang and Jenkins, 1993; Raabe and Roters, 2004). It should be remarked that in the glaciological literature the term "ODF" sometimes refers to the relative number, instead of the mass (or volume) fraction, of grains with a certain orientation.

The time evolution of ϱ^* is governed by the balance equation of species (orientational) mass

$$\frac{\partial \varrho^*}{\partial t} + \operatorname{div}(\varrho^* \boldsymbol{v}) + \operatorname{div}_{\boldsymbol{n}}(\varrho^* \boldsymbol{u}^*) = \varrho^* \Gamma^* \tag{39}$$

with

$$\begin{aligned} \operatorname{div}_{\boldsymbol{n}}(\boldsymbol{\Phi}^*) &= \operatorname{tr}\left[\operatorname{grad}_{\boldsymbol{n}}(\boldsymbol{\Phi}^*)\right], \\ \operatorname{grad}_{\boldsymbol{n}}(\boldsymbol{\Phi}^*) &= \frac{\partial \boldsymbol{\Phi}^*}{\partial \boldsymbol{n}} - \left(\frac{\partial \boldsymbol{\Phi}^*}{\partial \boldsymbol{n}} \cdot \boldsymbol{n}\right)\boldsymbol{n} \end{aligned} \tag{40}$$

for any scalar-, vector- or tensor-valued field $\boldsymbol{\Phi}^*(\boldsymbol{x}, t, \boldsymbol{n})$. In (39), $\boldsymbol{u}^*(\boldsymbol{x}, t, \boldsymbol{n})$ denotes a sort of "velocity" on the unit sphere (with $\boldsymbol{u}^* \cdot \boldsymbol{n} = 0$), called orientational transition rate. Further, $\Gamma^*(\boldsymbol{x}, t, \boldsymbol{n})$ is the specific recrystallization rate, which describes the rate of change of mass (per unit mass) of one species into another one with different orientation. Integration of (39) over the unit sphere $\mathcal{S}^2$ gives rise to the usual mass balance equation for the polycrystal (i.e. the "mixture")

$$\begin{aligned} &\frac{\partial \varrho}{\partial t} + \operatorname{div}(\varrho \boldsymbol{v}) = 0 \quad \text{with} \\ &\int_{\mathcal{S}^2} \varrho^* \Gamma^* \,\mathrm{d}^2n = \int_{\mathcal{S}^2} \operatorname{div}_{\boldsymbol{n}}(\varrho^* \boldsymbol{u}^*)\,\mathrm{d}^2n = 0\,, \end{aligned} \tag{41}$$

Notice that the first integral in (41) is a consequence of mass conservation, while the second integral follows from Gauss' theorem.

As shown by Faria (2001, 2006a) and Faria and Hutter (2002), the transition rate $\boldsymbol{u}^*$ is governed by its own balance equation, involving couple stresses and body couples. In the development of the CAFFE model, however, an abridged approach has been adopted by postulating a constitutive equation for the transition rate

$$\boldsymbol{u}^* = \boldsymbol{W}\boldsymbol{n} - \iota\left[\dot{\boldsymbol{\varepsilon}}\boldsymbol{n} - (\boldsymbol{n} \cdot \dot{\boldsymbol{\varepsilon}}\boldsymbol{n})\,\boldsymbol{n}\right] - \frac{\lambda}{\varrho^*} \operatorname{grad}_{\boldsymbol{n}}(\varrho^* H^*) \tag{42}$$

where

$$\boldsymbol{W} = \frac{1}{2}\left(\operatorname{grad}\boldsymbol{v} - (\operatorname{grad}\boldsymbol{v})^T\right), \dot{\boldsymbol{\varepsilon}} = \frac{1}{2}\left(\operatorname{grad}\boldsymbol{v} + (\operatorname{grad}\boldsymbol{v})^T\right) \tag{43}$$

are the tensors of rotation and strain rate, respectively. The first term on the right hand side of (42) represents a rigid-body rotation, while the second term describes the process of strain-induced lattice rotation (Dafalias, 2001), with $\iota > 0$ denoting the so-called "shape factor" of the theory of rotational diffusion (Faria, 2001). According to Placidi et al. (2010), fabric evolution simulations of the GRIP and EPICA-DML ice cores suggest that best results are obtained for $0.6 > \iota > 0.4$. Finally, the third term on the right hand side of (42) models rotation recrystallization as a diffusive process, with $\lambda > 0$ being the orientational diffusivity and $H^*(\boldsymbol{x}, t, \boldsymbol{n})$ an orientational ("chemical") potential, also called "hardness function" by Gödert (2003). In principle H^* should be a constitutive function, but, based on microstructural analyses of the NorthGRIP ice core (Durand et al., 2008), Placidi et al. (2010) suggest that one may simply set $H^* = 1$.

In the original application of the MCD theory to the flow of glaciers and ice sheets (Faria, 2006b), the specific recrystallization rate Γ^* is regarded as a dissipative variable. However, for simplicity, in the CAFFE model Placidi (2004, 2005) has proposed the following relation between Γ^* and the strain rate

$$\begin{aligned} &\Gamma^* = G\,(D^* - D)\,, \quad \text{with} \\ &D^* = 5\frac{(\dot{\boldsymbol{\varepsilon}}\boldsymbol{n})^2 - (\boldsymbol{n} \cdot \dot{\boldsymbol{\varepsilon}}\boldsymbol{n})^2}{\operatorname{tr}\left(\dot{\boldsymbol{\varepsilon}}^2\right)} \quad \text{and} \quad D = \frac{1}{\varrho}\int_{\mathcal{S}^2} \varrho^* D^* \mathrm{d}^2n\,, \end{aligned} \tag{44}$$

where $G > 0$ is a material parameter, while $5/2 \geq D^* \geq 0$ and $5/2 \geq D \geq 0$ are called the species and polycrystal "deformability", respectively.

As remarked by Placidi et al. (2010), owing to the difficulties in determining the values of the material parameters λ and G from experiments, they are usually determined by fitting numerical simulations of ice core fabrics and grain stereology. This concludes the description of the fabric evolution.

As for the flow law, in contrast to the full stress–strain rate relation with tensorial fluidity (viscosity) predicted by the theory of continuous diversity (Faria, 2006b), the CAFFE model adopts a much simplified generalization of Glen's flow law:

$$\dot{\boldsymbol{\varepsilon}} = E(D)\,A(T)\,\sigma_{eq}^{n-1}\,\boldsymbol{\sigma}'\,, \tag{45}$$

where $\boldsymbol{\sigma}'$ is the deviatoric part of the Cauchy stress tensor $\boldsymbol{\sigma}$, σ_{eq} is the effective stress invariant, n is the power law exponent (usually set equal 3), T is the temperature, and $A(T)$ is a temperature-dependent rate factor. Clearly, (45) implies that all anisotropy effects are contained in the scalar-valued, deformability-dependent flow enhancement factor $E(D)$, such that stress and strain rate are collinear and (45) reduces to the classical form of Glen's flow law when $E(D) \equiv \text{const}$.

A detailed functional form for the enhancement factor $E(D)$ has been proposed by Seddik et al. (2008) and Placidi et al. (2010), which is continuously differentiable at $D = 1$ and is compatible with the experimental results of Azuma (1995) and Miyamoto (1999)

$$E(D) = \begin{cases} (1 - E_{\min})\,D^{\zeta} + E_{\min} & 1 \geq D \geq 0\,, \\ \dfrac{4D^2\,(E_{\max} - 1) + 25 - 4E_{\max}}{21} & 5/2 \geq D > 1\,, \end{cases} \tag{46}$$

with

$$\zeta = \frac{8}{21}\left(\frac{E_{\max} - 1}{1 - E_{\min}}\right), \qquad E_{\max} \approx 10\,, \quad E_{\min} \approx 0.1\,. \tag{47}$$

By introducing the orientation tensors (essentially equivalent to the dipole and quadrupole moments of ϱ^*/ϱ)

$$\boldsymbol{a}^{(2)} = \frac{1}{\varrho}\int_{\mathcal{S}^2} \varrho^* \boldsymbol{n}\otimes\boldsymbol{n}\,\mathrm{d}^2 n\ , \quad \boldsymbol{a}^{(4)} = \frac{1}{\varrho}\int_{\mathcal{S}^2} \varrho^* \boldsymbol{n}\otimes\boldsymbol{n}\otimes\boldsymbol{n}\otimes\boldsymbol{n}\,\mathrm{d}^2 n \tag{48}$$

to reformulate the CAFFE flow law (45) in an explicitly anisotropic form

$$\dot{\boldsymbol{\varepsilon}} = \hat{E}(\boldsymbol{\sigma}')\,A(T)\,\sigma_{eq}^{n-1}\,\boldsymbol{\tau}\ , \tag{49}$$

In plain words, (49) tells us that the CAFFE model can be applied to all anisotropies (fabrics) that can satisfactorily be represented by a multipole expansion up to fourth order. Fortunately, most anisotropies observed in glaciers and ice sheets.

6.2. GOLF law and Elmer/Ice

In this section, we present the anisotropic ice flow model developed at LGGE. This model has been used for various applications (Gillet-Chaulet et al., 2005, 2006; Durand et al., 2007; Martín et al., 2009; Ma et al., 2010). In this approach, the fabric is described using the second and fourth-order orientation tensors (48). In this continuum description of the fabric, the polycrystal represents the local behavior of a representative elementary ice volume. By assuming that the fourth-order orientation tensor $\boldsymbol{a}^{(4)}$ is given as a tensorial function of $\boldsymbol{a}^{(2)}$ (Gillet-Chaulet et al., 2005), the fabric can be described in a very condensed way using $\boldsymbol{a}^{(2)}$ solely. By definition, $\mathrm{tr}\,\boldsymbol{a}^{(2)} = 1$, so that only the first two eigenvalues $a_1^{(2)}$ and $a_2^{(2)}$ and three Euler angles are needed to completely define the fabric. As a consequence, modeled fabrics are orthotropic, *i.e.* the c-axes distribution presents three orthogonal symmetry planes. Although orthotropy is a simple form of the most general anisotropy, it is thought to be a good compromise between physical adequateness and simplicity. The second-order orientation tensor allows to describe all the observed fabric patterns: for random c-axes distribution the diagonal entries of $\boldsymbol{a}^{(2)}$ are $a_{11}^{(2)} = a_{22}^{(2)} = a_{33}^{(2)} = 1/3$, for a single maximum fabric with its maximum in the third direction, $a_{33}^{(2)} > 1/3$ and $a_{11}^{(2)} \approx a_{22}^{(2)} < 1/3$, and for a girdle type fabric in the plane (x_1, x_2), $a_{33}^{(2)} < 1/3$ and $a_{11}^{(2)} \approx a_{22}^{(2)} > 1/3$. When the material symmetry axes are those of the general reference frame, as for the three particular previous fabrics, the non-diagonal entries of $\boldsymbol{a}^{(2)}$ are zero.

The behavior of the polycrystal is described by the general orthotropic linear flow law (GOLF, Gillet-Chaulet et al., 2005). In its initial form, ice was assumed to behave as a linearly viscous orthotropic material. In more recent works (Martín et al., 2009; Ma et al., 2010), the GOLF law has been extended to a nonlinear form by adding an invariant in the anisotropic linear law. The simple choice is either to add the second invariant of the strain rate (Martín et al., 2009) or the second invariant of the deviatoric stress (Pettit et al., 2007). No theoretical or experimental results are available today to discard one of these two solutions, and other solutions based on anisotropic invariants of the deviatoric stress and/or the strain rate are also possible. In (Ma et al., 2010) approach, the nonlinearity of the law is introduced through the second invariant of the deviatoric stress. With this definition, the anisotropy factors of the polycrystalline law for a given stress are identical in the linear and nonlinear cases. In other words, for a given fabric and a given state of stress, the corresponding strain rate relative to the isotropic response is the same for the linear and nonlinear cases. Using the strain-rate invariant in the same way as Martín et al. (2009) did, leads to different anisotropy factors (as defined here) in the linear and nonlinear cases. Therefore, the proposed expression of the nonlinear GOLF law is as follows:

$$\sum_{r=1}^{3}\left[\eta_r tr(\boldsymbol{M}_r\cdot\dot{\boldsymbol{\varepsilon}})\boldsymbol{M}_r' + \eta_{r+3}(\dot{\boldsymbol{\varepsilon}}\cdot\boldsymbol{M}_r + \boldsymbol{M}_r\cdot\dot{\boldsymbol{\varepsilon}})'\right] = 2A\sigma_{eq}^{n-1}\boldsymbol{\sigma}'\ , \tag{50}$$

where A is the temperature-dependent Glen's law parameter for isotropic ice. The six dimensionless anisotropy viscosities $\eta_r(\boldsymbol{a}^{(2)})$ and $\eta_{r+3}(\boldsymbol{a}^{(2)})$ (r =1, 2, 3) are functions of eigenvalues of the second-order orientation tensor $\boldsymbol{a}^{(2)}$, which represent a measure of the anisotropy strength. The three structure tensors $\boldsymbol{M}_r$ are given by the dyadic products of the three eigenvectors of $\boldsymbol{a}^{(2)}$, which then represent the material symmetry axes. In the method proposed by Gillet-Chaulet et al. (2005), the six dimensionless viscosities $\eta_r(\boldsymbol{a}^{(2)})$ are tabulated as a function of the fabric strength (*i.e.*, the $a_i^{(2)}$) using a micro-macro model. When ice is isotropic, $\eta_r = 0$ and $\eta_{r+3} = 1$ (r =1, 2, 3), and Eq. (50) reduces to the isotropic Glen's flow law.

Following Gillet-Chaulet et al. (2005), the six dimensionless viscosities $\eta_r(\boldsymbol{a}^{(2)})$ are tabulated using the visco-plastic self-consistent model (VPSC, Castelnau et al., 1996a, 1998), see Section 3. The two crystal parameters in the VPSC model used to tabulate the GOLF law were chosen so that the experimentally observed polycrystal anisotropy is reproduced. Gillet-Chaulet et al. (2005) use the shear-strain rates ratio for a polycrystal with a single maximum fabric and an isotropic polycrystal both experiencing the same shear stress. This anisotropy factor in shear is hereafter noted k_s and, according to the experimental results of Pimienta et al. (1987), its value is approximately $k_s = 10$. In other words, the VPSC parameters are chosen so that the response under simple shear of a polycrystal with a single maximum fabric is k_s times easier to deform than the corresponding isotropic polycrystal. The experimental results of Pimienta et al. (1987) also indicate that an isotropic polycrystal is much easier to deform than a single maximum fabric polycrystal experiencing the same uniaxial compressional stress. These experiments allow to define a second anisotropy factor for uniaxial compressional stress, which is noted k_c. A value $k_c = 0.4$ is in accordance with the experimental results of Pimienta et al. (1987). As discussed before, the anisotropy factors k_s and k_c are independent of Glen's flow law exponent n with the adopted nonlinear formulation.

Assuming that recrystallization processes do not occur and that the ice fabric is induced solely by deformation, the evolution of the second-order orientation tensor $\boldsymbol{a}^{(2)}$ can be written as

$$\frac{D\boldsymbol{a}^{(2)}}{Dt} = \boldsymbol{W}\cdot\boldsymbol{a}^{(2)} - \boldsymbol{a}^{(2)}\cdot\boldsymbol{W} - (\boldsymbol{C}\cdot\boldsymbol{a}^{(2)} + \boldsymbol{a}^{(2)}\cdot\boldsymbol{C}) + 2\boldsymbol{a}^{(4)} : \boldsymbol{C}\ , \tag{51}$$

where $\boldsymbol{W}$ is the spin tensor defined as the antisymmetric part of

the velocity gradient. The tensor $\boldsymbol{C}$ is defined as

$$\boldsymbol{C} = (1-\alpha)\dot{\boldsymbol{\varepsilon}} + \alpha\, k_s\, A\, \sigma_{eq}^{n-1}\, \boldsymbol{\sigma}' \,. \tag{52}$$

The *interaction parameter* α controls the relative weighting of the strain rate $\dot{\boldsymbol{\varepsilon}}$ and the deviatoric stress $\boldsymbol{\sigma}'$ in the fabric evolution Eq. (51). When $\alpha = 0$, the fabric evolution is solely controlled by the state of strain rate, whereas in the case where $\alpha = 1$ the fabric evolves under the influence of the deviatoric stress solely. In between, as for the VPSC, both the strain rate and deviatoric stress contribute to the fabric evolution. In what follows, the interaction parameter is $\alpha = 0.06$ in accordance with the crystal anisotropy and the VPSC model used to derive the polycrystal behaviour (Gillet-Chaulet et al., 2005). In Eq. (51), the fourth-order orientation tensor is evaluated assuming a closure approximation giving $\boldsymbol{a}^{(4)}$ as a tensorial function of $\boldsymbol{a}^{(2)}$ (Gillet-Chaulet et al., 2005).

The anisotropic polycrystalline law described above and the associated fabric evolution equations have been implemented in the Finite Element code Elmer/Ice, the glaciological part of the open source Finite Element software Elmer developed by CSC (http://www.elmerfem.org/). Ice flow (velocity and isotropic pressure) are obtained solving the anisotropic Stokes equations and coupled with the fabric evolution equation (51) and the upper free surface equation in the case of transient simulations. In Gillet-Chaulet et al. (2006), the model was applied to synthetic geometries in order to show the influence of coupling the Stokes and fabric evolution equations on the flow of ice over a bumpy bedrock. In Durand et al. (2007), the model was used to explain the fabric evolution in the Dome C ice core, in the framework of the EPICA project. The authors showed that to explain the fabric evolution at Dome C, shear stress must be invoked. The model was also applied to evaluate the value of the ad-hoc enhancement factor that should be incorporated in large-scale isotropic ice-sheet flow model in Ma et al. (2010). In Martín et al. (2009), the anisotropic ice flow model was applied to explain observed shapes of isochrones below ridges or domes.

7. Synthesis and perspectives

Applications of ice mechanical behavior modeling extend from below the single-crystal scale to the ice sheet scale. Upwards, this scale range far exceeds that of engineering material sciences but is similar to the geological one. Within this scale range, many physical processes come into play, some of which are not yet very well described. Furthermore, there exist strong interactions between these processes that create bridges between the different levels of complexity. Modeling of ice has strongly benefited from advances in materials science. In return, as shown by the results presented in this paper, the contribution of the ice community to the theoretical understanding and modeling of the mechanical behavior of anisotropic materials is significant. With the large viscoplastic anisotropy of the ice crystal, ice is now considered a model material. The advances presented here may equally well be applied to, for example, mantle flow, where the anisotropy due to fabric (CPO) development in olivine is thought to play a significant role (Tommasi et al., 2009; Long and Becker, 2010).

The presented modeling methods are basically of two types; some that aim to precisely reproduce the physical mechanisms as observed experimentally, and some with a more phenomenological approach. Going through scales, it clearly appears that individual dislocation interactions cannot be taken into account at the scale of the polycrystal imbedded in a glacier environment. Nevertheless, modeling at the scale of dislocation interactions provides a better estimate of the interactions between slip systems at the single crystal scale, which, in turn, is essential to reproduce an accurate mechanical response of the polycrystal with mean-field and full-field approaches. Furthermore, full-field approaches are necessary to validate the approximations made using mean-field models, as they provide the "exact" (in a numerical sense) response of the specimen with a real microstructure, integrating the inter- and intra-granular interactions. Finally, large-scale flow models are now getting to a sufficient level of complexity to be able to take into account and represent the anisotropy associated with the fabrics induced by the flow conditions. To do so, they integrate mean-field approaches that correctly reproduce the viscoplastic anisotropy and a nonlinear mechanical behavior.

A summary of the main domains of application, advantages, and limitations of the main modeling tools presented in the paper is given on tables 2, 3 and 4

Much progress has recently been made in the modeling of dynamic recrystallization processes and their interactions with flow anisotropy. Nevertheless, due to the complexity of the physical processes involved, to jump the gap between scales is a strong challenge. The field dislocation mechanics approach appears very promising to associate the internal stress field and dislocation arrangements to the nucleation and grain boundary migration mechanisms. However, field dislocation mechanics cannot yet be applied to scales larger than the polycrystal. Full-field models, including the FFT-Elle coupling, have the same scale limitation, but may play an important role in parameterizing small-scale processes (dislocation glide, grain boundary migration, etc.) for mean-field models. They are also important tools to test models of mechanical and microstructural evolution.

Compared to other minerals, ice shows remarkably strong transient behavior (Duval et al., 1983; Castelnau et al., 2008b). Continuum flow models, such as Glen law (Glen, 1955) have so far not been able to incorporate the resulting mechanical complexity of polycrystalline ice deformation. Only recently have mechanical models reached a level of sophistication to address transient behavior. This development is promising and probably highly relevant in cases where ice flow changes at rates for where both elastic and viscoplastic behavior may interact. In particular, this concerns the very topical subject of ice shelves, ice streams or extra-terrestrial ice submitted to tide forcing. Which model will be able to correctly take into account these transient, and event cyclic behavior, and at which scale?

A next step will likely be the multi-scale coupling of models of increasing complexity. We can expect dislocation dynamics and field dislocation mechanics to provide the local criteria for

slip system interactions, nucleation, grain boundary migration as local input to full-field approaches that will be further used in interaction with mean-field approaches to calibrate dynamic recrystallization variables influencing the mechanical response and fabric development.

An interesting example of such model interweaving is given by the large-scale flow modeling presented in this paper. Nevertheless, a strong effort is still required concerning the flow law of ice and its dependency on fabrics (CPO) and strain. Recent velocity measurements in Greenland (Gillet-Chaulet et al., 2011) questioned the relevance of a stress exponent equal to three as classically considered for large scale flow modeling (for instance Paterson (1994); Hooke (2005); Greve and Blatter (2009), ...). Owing to the variety of processes that accommodate strain along an ice core path, one could also expect several regimes to occur with depth, as suggested by some authors (see for instance Lipenkov et al. (1989); Faria et al. (2009); Pettit et al. (2011)). Such modeling - observation comparisons mainly raise the complexity of the physical processes involved that can probably not be summarized in a single universal law.

8. Acknowledgement

Financial support by the French "Agence Nationale de la Recherche" is acknowledged (project ELVIS, #ANR-08-BLAN-0138). Together with support from institutes INSIS and INSU of CNRS, and UJF - Grenoble 1, France. PDB and JR gratefully acknowledge funding by the German Research Foundation (DFG, project BO-1776/7). The authors gratefully aknowledge the ESF Research Networking Programme Micro-Dynamics of Ice (MicroDIce).

References

Acharya, A.. A model of crystal plasticity based on the theory of continuously distributed dislocations. Journal of the Mechanics and Physics of Solids 2001;49(4):761 – 784.

Acharya, A., Roy, A.. Size effects and idealized dislocation microstructure at small scales: Predictions of a phenomenological model of mesoscopic field dislocation mechanics: Part I. Journal of the Mechanics and Physics of Solids 2006;54(8):1687 – 1710.

Alley, R.B.. Fabrics in polar ice sheets - Development and prediction. Science 1988;240:493–495.

Alley, R.B.. Flow-law hypotheses for ice-sheet modeling. J Glaciol 1992;38(129):245–255.

Alley, R.B., Perepezko, J.H., Bentley, C.R.. Grain growth in polar ice: I. theory. J Glaciol 1986;32(112):415–424.

Arsenlis, A., Parks, D.M.. Crystallographic aspects of geometrically-necessary and statistically-stored dislocation density. Acta Materialia 1999;47:1597–1611.

Ashby, M.F., Duval, P.. the creep of polycrystalline ice. Cold Reg Sc Tech 1985;11:285–300.

Azuma, N.. A flow law for anisotropic ice and its application to ice sheets. Earth and Planetary Science Letters 1994;128(3â"4):601 – 614.

Azuma, N.. A flow law for anisotropic polycrystalline ice under uniaxial compressive deformation. Cold Reg Sci Technol 1995;23:137–147.

Azuma, N., Higashi, A.. Formation processes of ice fabric pattern in ice sheets. Ann Glaciol 1985;6:130–134.

Bargmann, S., Seddik, H., Greve, R.. Computational modeling of flow-induced anisotropy of polar ice for the EDML deep drilling site, Antarctica: the effect of rotation recrystallization and grain boundary migration. Int J Numer Anal Meth Geomech 2011;:DOI: 10.1002/nag.1034.

Barnes, P., Tabor, D., Walker, J.. The friction and creep of polycrystalline ice. Proceeding of the Royal Society of London Series A, Mathematical and Physical Sciences 1971;324(1557):127–155.

Barr, T., Houseman, G.. Deformation fields around a fault embedded in a non-linear ductile medium. Geophysical Journal International 1996;125:473–490.

Battaile, C., Counts, W., Wellman, G., Buchheit, T., Holm, E.. Simulating grain growth in a deformed polycrystal by coupled finite-element and microstructure evolution modeling. Metallurgical and Materials Transactions A 2007;38:2513–2522. 10.1007/s11661-007-9267-6.

Becker, J.K., Bons, P.D., Jessell, M.W.. A new front-tracking method to model anisotropic grain and phase boundary motion in rocks. Computers & Geosciences 2008;34:201–212.

Bobeth, M., Diener, G.. Static and thermoelastic field fluctuations in multiphase composites. JMech Phys Solids 1987;35:137–149.

Boehler, J.P., Aoufi, L.E., Raclin, J.. On experimental testing methods for anisotropic materials. Res Mech 1987;21:73–95.

Bons, P.D., Barr, T.D., ten Brink, C.E.. The development of delta-clasts in non-linear viscous materials: a numerical approach. Tectonophysics 1997;270:29–41.

Bons, P.D., Jessell, M.W., Evans, L., Barr, T.D., Stüwe, K.. Modelling of anisotropic grain growth in minerals. Geological Society of America Memoir 2001;193:39–49.

Bons, P.D., Koehn, D., Jessell, M.W.. Lecture notes in earth sciences. In: Bons, P., Koehn, D., Jessell, M., editors. Microdynamic Simulation. Springer, Berlin; number 106; 2008. 405pp.

Bornert, M., Masson, R., Ponte Castañeda, P., Zaoui, A.. Second-order estimates for the effective behaviour of viscoplastic polycrystalline materials. J Mech Phys Solids 2001;49:2737–2764.

Bornert, M., Ponte Castañeda, P.. Second-order estimates of the self-consistent type for viscoplastic polycrystals. ProcRSocLond 1998;A454:3035–3045.

Bouchez, J.L., Duval, P.. The fabric of polycrystalline ice deformed in simple shear : experiments in torsion, natural deformation and geometrical interpretation. Textures and microstructures 1982;5:171–190.

Brenner, R., Béchade, J.L., Castelnau, O., Bacroix, B.. Thermal creep of Zr-Nb1%-O alloys: experimental analysis and micromechanical modelling. J Nucl Mater 2002a;305:175–186.

Brenner, R., Castelnau, O., Badea, L.. Mechanical field fluctuations in polycrystals estimated by homogenization techniques. ProcR SocLond 2004;A460(2052):3589–3612.

Brenner, R., Lebensohn, R.A., Castelnau, O.. Elastic anisotropy and yield surface estimates. Int J Solids Struct 2009;46:3018–3026.

Brenner, R., Masson, R., Castelnau, O., Zaoui, A.. A "quasi-elastic" affine formulation for the homogenized behaviour of nonlinear viscoelastic polycrystals and composites. Eur J Mech A/Solids 2002b;21:943–960.

Brinckmann, S., Siegmund, T., Huang, Y.. A dislocation density based strain gradient model. International Journal of Plasticity 2006;22:1784–1797.

Budd, W., Jacka, T.. A review of ice rheology for ice sheet modelling. Cold Reg Sci Technol 1989;16:107–144.

Buiron, D., Chappellaz, J., Stenni, B., Frezzoti, M., Baumgartner, M., Capron, E., Landais, A., Lemieux-Dudon, B., Masson-Delmotte, V., Montagnat, M., Parrenin, F., Schilt, A.. TALDICE-1 age scale of the Talos Dome deep ice core, East Antarctica. Climate of the Past 2011;7:1–16.

Bunge, H.J.. Texture Analysis in Materials Science. 3rd ed. Goettingen: Cuvillier, 1993.

Castelnau, O., Blackman, D.K., Becker, T.W.. Numerical simulations of texture development and associated rheological anisotropy in regions of complex mantle flow. Geophys Res Let 2009;36(L12304).

Castelnau, O., Blackman, D.K., Lebensohn, R.A., Ponte-Castañeda, P.. Micromechanical modeling of the viscoplastic behavior of olivine. Journal of Geophysical Research 2008a;113:B09202.

Castelnau, O., Brenner, R., Lebensohn, R.A.. The effect of strain heterogeneity on the work-hardening of polycrystals predicted by mean-field approaches. Acta Materialia 2006;54:2745–2756.

Castelnau, O., Canova, G.R., Lebensohn, R.A., Duval, P.. Modelling viscoplastic behavior of anisotropic polycrystalline ice with a self-consistent approach. Acta Materialia 1997;45(11):4823 – 4834.

Castelnau, O., Cordier, P., Lebensohn, R.A., Merkel, S., Raterron, P.. Microstructures and rheology of the earth's upper mantle inferred from a multiscale approach. Comptes Rendus Physique 2010a;11(3-4):304 – 315. Computational metallurgy and scale transitions.

Castelnau, O., Duval, P.. Simulations of anisotropy and fabric development in polar ices. Ann Glaciol 1994;20:277–282.

Castelnau, O., Duval, P., Lebensohn, R.A., Canova, G.. Viscoplastic modeling of texture development in polycrystalline ice with a self-consistent approach : Comparison with bound estimates. J Geophys Res 1996a;101(6):13,851–13,868.

Castelnau, O., Duval, P., Montagnat, M., Brenner, R.. Elastoviscoplastic micromechanical modeling of the transient creep of ice. Journal of Geophysical Research Solid Earth 2008b;113(B11203).

Castelnau, O., Lebensohn, R.A., Ponte Castañeda, P., Blackman, D.. Earth Mantle Rheology Inferred from Homogenization Theories; ISTE. p. 55–70.

Castelnau, O., Shoji, H., Mangeney, A., Milsch, H., Duval, P., Miyamoto, A., Kawada, K., Watanabe, O.. Anisotropic behavior of GRIP ices and flow in Central Greenland. Earth and Planetary Science Letters 1998;154(1-4):307 – 322.

Castelnau, O., Thorsteinsson, T., Kipfstuhl, J., Duval, P., Canova, G.R.. Modelling fabric development along the GRIP ice core, central Greenland. Ann Glaciol 1996b;23:194–201.

Chaboche, J.L.. A review of some plasticity and viscoplasticity constitutive theories. International Journal of Plasticiy 2008;24:1642–1693.

de la Chapelle, S., Castelnau, O., Lipenkov, V., Duval, P.. Dynamic recrystallization and texture development in ice as revealed by the study of deep ice cores in Antarctica and Greenland. J Geophys Res 1998;103(B3):5091–5105.

Chevy, J., Fressengeas, C., Lebyodkin, M., Taupin, V., Bastie, P., Duval, P.. Characterizing short-range vs. long-range spatial correlations in dislocation distributions. Acta Materialia 2010;58(5):1837 – 1849.

Chevy, J., Louchet, F., Duval, P., Fivel, M.. Creep behaviour of ice single crystals loaded in torsion explained by dislocation cross-slip. Phil Mag Let 2012;92(6):262–269. In press.

Chevy, J., Montagnat, M., Duval, P., Fivel, M., Weiss, J.. Dislocation patterning and deformation processes in ice single crystal deformed by torsion. Proc 11th Int Conf Phys & Chem of Ice 2007;:142–146.

Cochard, J., Yonenaga, I., Gouttebroze, S., MHamdi, M., Zhang, Z.L.. Constitutive modeling of intrinsic silicon monocrystals in easy glide. Journal of Applied Physics 2010;107(3):033512 –033512–9.

Dafalias, Y.F.. Orientation distribution function in non-affine rotations. J Mech Phys Solids 2001;49:2493–2516.

Dahl-Jensen, D., Gundestrup, N.S.. Constitutive properties of ice at Dye 3, Greenland. In: The physical basis of ice sheet modelling. Vancouver Symposium; IAHS; volume 170; 1987. p. 31–43.

Dansgaard, W., Johnsen, S.J., Clausen, H.B., Dahl-Jensen, D., Gundestrup, N.S., Hammer, C.U., Hvidberg, C.S., Steffensen, J.P., Sveinbjörnsdottir, A.E., Jouzel, J., Bond, G.. Evidence for general instability of past climate from a 250 kyr ice-core record. Nature 1993;364:218–220.

De Botton, G., Ponte Castañeda, P.. Variational estimates for the creep behaviour of polycrystals. Proc R Soc Lond 1995;A 448:121–142.

Durand, G., Gillet-Chaulet, F., Svensson, A., Gagliardini, O., Kipfstuhl, S., Meyssonnier, J., Parrenin, F., Duval, P., Dahl-Jensen, D., Azuma, N.. Change of the ice rheology with climatic transitions. Implication on ice flow modelling and dating of the EPICA Dome C core. Climates of the Past 2007;3:155–167.

Durand, G., Graner, F., Weiss, J.. Deformation of grain boundaries in polar ice. EPL (Europhysics Letters) 2004;67(6):1038.

Durand, G., Persson, A., Samyn, D., Svensson, A.. Relation between neighbouring grains in the upper part of the NorthGRIP ice core - Implications for rotation recrystallization. Earth and Planet Sc Let 2008;265:666–671.

Duval, P.. Lois du fluage transitoire ou permanent de la glace polycristalline pour divers états de contraintes. Ann Geophys 1976;32(4):335–350.

Duval, P.. Anelastic behaviour of polycrystalline ice. J Glaciol 1978;21(85):621–628.

Duval, P.. Creep and fabrics of polycrystalline ice under shear and compression. J Glaciol 1981;27(95):129–140.

Duval, P., Arnaud, L., Brissaud, O., Montagnat, M., de La Chapelle, S.. Deformation and recrystallization processes of ice from polar ice sheets. Ann Glaciol 2000;30:83–87.

Duval, P., Ashby, M., Anderman, I.. Rate controlling processes in the creep of polycrystalline ice. J Phys Chem 1983;87(21):4066–4074.

Duval, P., Castelnau, O.. Dynamic recrystallization of ice in polar ice sheets. J Physique IV (suppl J Phys III), C3 1995;5:197–205.

Duval, P., Le Gac, H.. Does the permanent creep-rate of polycrystalline ice increase with crystal size? Journal of Glaciology 1980;25:151–157.

Duval, P., Montagnat, M.. Comment on "Superplastic deformation of ice: experimental observations" by D.L. Goldsby and D.L. Kohlstedt. J Geophys Res 2002;107:(2082)1–4.

Duval, P., Montagnat, M., Grennerat, F., Weiss, J., Meyssonnier, J., Philip, A.. Creep and plasticity of glacier ice: a material science perspective. Journal of Glaciology 2010;56(200):1059–1068.

Eshelby, J.. The determination of the elastic field of an ellipsoidal inclusion, and related problems. Proc R Soc London Ser A 1957;241:376–396.

Faria, S.H.. Mixtures with continuous diversity: general theory and application to polymer solutions. Continuum Mechanics and Thermodynamics 2001;13(2):91–120.

Faria, S.H.. Creep and recrystallization of large polycrystalline masses. I. General continuum theory. Royal Society of London Proceedings Series A 2006a;462(2069):1493–1514.

Faria, S.H.. Creep and recrystallization of large polycrystalline masses. III: Continuum theory of ice sheets. Royal Society of London Proceedings Series A 2006b;462:2797–2816.

Faria, S.H., Freitag, J., Kipfstuhl, S.. Polar ice structure and the integrity of ice-core paleoclimate records. Quat Sci Rev 2010;29(1):338–351.

Faria, S.H., Hutter, K.. A systematic approach to the thermodynamics of single and mixed flowing media withmicrostructure. Part I: balance equations and jump conditions. Continuum Mech Thermodyn 2002;14(5):459–481.

Faria, S.H., Kipfstuhl, S., Azuma, N., Freitag, J., Hamann, I., Murshed, M.M., Kuhs, W.F.. The multiscale structure of Antarctica. Part I: inland ice. Low Temp Sci 2009;68:39–59.

Faria, S.H., Kremer, G.M., Hutter, K.. On the inclusion of recrystallization processes in the modeling of induced anisotropy in ice sheets: a thermodynamicist's point of view. Ann Glaciol 2003;37(1):29–34.

Faria, S.H., Kremer, G.M., Hutter, K.. Creep and recrystallization of large polycrystalline masses. II. Constitutive theory for crystalline media with transversely isotropic grains. Royal Society of London Proceedings Series A 2006;462(2070):1699–1720.

Faria, S.H., Ktitarev, D., Hutter, K.. Modelling evolution of anisotropy in fabric and texture of polar ice. Ann Glaciol 2002;35:545–551.

Faria, S.H., Weikusat, I., Azuma, N.. The microstructure of polar ice. J Struct Geol this issue;.

Fressengeas, C.. La mécanique des champs de dislocations. Hermés Sciences, 2010.

Gagliardini, O., Gillet-Chaulet, F., Montagnat, M.. A review of anisotropic polar ice models: from crystal to ice-sheet flow models. In: Hondoh, T., editor. Physics of Ice Core Records II. Supplement Issue of Low Temperature Science, Hokkaido University; volume 68; 2009. p. 149–166.

Gammon, P., Kiefte, H., Clouter, M.. Elastic constants of ice samples by Brillouin spectroscopy. J Phys Chem 1983;87:4025–4029.

Gao, H., Huang, Y., Nix, W.D., Hutchinson, J.W.. Mechanism-based strain gradient plasticity – I. theory. Journal of the Mechanics and Physics of Solids 1999;47(6):1239 – 1263.

Gillet, F., Durand, G.. Ice-sheet advance in Antarctica. Nature 2010;467:794–795.

Gillet-Chaulet, F., Gagliardini, O., Meyssonnier, J., Montagnat, M., Castelnau, O.. A user-friendly anisotropic flow law for ice-sheet modelling. J Glaciol 2005;41(172):3 – 14.

Gillet-Chaulet, F., Gagliardini, O., Meyssonnier, J., Zwinger, T., Ruokolainen, J.. Flow-induced anisotropy in polar ice and related ice-sheet flow modelling. J Non-Newtonian Fluid Mech 2006;134:33–43.

Gillet-Chaulet, F., Gagliardini, O., Seddik, H., Nodet, M., Durand, G., Ritz, C., Zwinger, T., Greve, R., Vaughan, D.G.. Greenland ice sheet contribution to sea-level rise from a new-generation ice-sheet model. The Cryosphere 2012;6(6):1561–1576.

Gillet-Chaulet, F., Hindmarsh, R.C.A., Corr, H.F.J., King, E.C., Jenkins, A.. In-situ quantification of ice rheology and direct measurement of the Raymond effect at Summit, Greenland using a phase-sensitive radar. Geophys Res Lett 2011;38(24).

Gilormini, P.. A critical evaluation for various non-linear extensions of the self-consistent model. In: Pineau, A., Zaoui, A., editors. IUTAM Symp. on Michromecanics of Plasticity and Damage of Multiphase Materials. Klower Acad. Publ., Sèvres, France; 1995. p. 67–74.

Glen, J.. The creep of polycrystalline ice. Proc Roy Soc London 1955;A228:519–538.

Gödert, G.. A mesoscopic approach for modelling texture evolution of polar

ice including recrystallization phenomena. Ann Glaciol 2003;37:23–28.
Goldsby, D.L., Kohlstedt, D.L.. Grain boundary sliding in fine-grained ice I. Scripta Mater 1997;37(9):1399–1406.
Gow, A.. On the rate of growth of grains and crystals in south polar firn. J Glaciol 1969;8:241–252.
Grennerat, F., Montagnat, M., Castelnau, O., Vacher, P., Moulinec, H., Suquet, P., Duval, P.. Experimental characterization of the intragranular strain field in columnar ice during transient creep. Acta Materialia 2012;60(8):3655–3666.
Greve, R., Blatter, H.. Dynamics of Ice Sheets and Glaciers. Berlin: Springer, 2009.
Griera, A., Bons, P.D., Jessell, M.W., Lebensohn, R.A., Evans, L., Gomez-Rivas, E.. Strain localization and porphyroclast rotation. Geology 2011;39:275–278.
Griera, A., Llorens, M.G., Gomez-Rivas, E., Bons, P.D., Jessell, M.W., Evans, L.A., Lebensohn, R.A.. Numerical modelling of porphyroclast and porphyroblast rotation in anisotropic rocks. Tectonophysics 2012;587:4–29. In press.
Guillopé, M., Poirier, J.. Dynamic recrystallization during creep of single-crystalline halite: an experimental study. J Geophys Res 1979;84:5557–5567.
Gundestrup, N., Hansen, B.L.. Bore-hole survey at Dye 3, South Greenland. Journal of Glaciology 1984;30:282–288.
Hamman, I., Weikusat, C., Azuma, N., Kipfstuhl, S.. Evolution of crystal microstructure during creep experiments. J Glaciol 2007;53(182):479–489.
Hershey, A.V.. The elasticity of an isotropic aggregate of anisotropic cubic crystals. J Appl Mech 1954;21:236–240.
Higashi, A., Koinuma, S., Mae, S.. Bending creep of ice single crystals. Jpn J Appl Phys 1965;4:575–582.
Hooke, R.L.. Principles of Glacier Mechanics. 2nd ed. Cambridge: Cambridge University Press, 2005.
Humphreys, F.J., Hatherly, M.. Recrystallization and related annealing phenomena. Second ed. Pergamon, Oxford, 2004.
Hutchinson, J.. Creep and plasticity of hexagonal polycrystals as related to single crystal slip. Metall Trans 1977;8A(9):1465–1469.
Idiart, M.I., Moulinec, H., Ponte Castañeda, P., Suquet, P.. Macroscopic behavior and field fluctuations in viscoplastic composites: Second-order estimates versus full-field simulations. Journal of the Mechanics and Physics of Solids 2006;54(5):1029 – 1063.
Jacka, T.. Investigations of discrepancies between laboratory studies on the flow of ice: density, sample shape and size and grain size. Annals of Glaciology 1994;19:146–154.
Jacka, T.H., Li, J.. The steady-state crystal size of deforming ice. Ann Glaciol 1994;20:13–18.
Jacka, T.H., Maccagnan, M.. Ice crystallographic and strain rate changes with strain in compression and extension. Cold Reg Sci Technol 1984;8:269–286.
Jessell, M.W.. Simulation of fabric development in recrystallizing aggregates. 1. Description of the models. Journal of Structural Geology 1988a;10:771–778.
Jessell, M.W.. Simulation of fabric development in recrystallizing aggregates. 2. Example model runs. Journal of Structural Geology 1988b;10:779–793.
Jessell, M.W., Bons, , P. D. Evans, L., Barr, T., Stüwe, K.. Elle: the numerical simulation of metamorphic and deformation microstructures. Computers & Geosciences 2001;27:17–30.
Jessell, M.W., Bons, P.D.. The numerical simulation of microstructure. Geol Soc, London, Spec Publ 2002;200:137–147.
Jessell, M.W., Bons, P.D., Griera, A., Evans, L.A., Wilson, C.J.L.. A tale of two viscosities. Journal of Structural Geology 2009;31:719–736.
Jessell, M.W., Siebert, E., Bons, P.D., Evans, L., Piazolo, S.. A new type of numerical experiment on the spatial and temporal patterns of localization of deformation in a material with a coupling of grain size and rheology. Earth and Planetary Science Letters 2005;239:309–326.
Jones, S., Glen, J.. The mechanical properties of single crystals of pure ice. J Glaciol 1969;8(54):463–473.
Kanit, T., Forest, S., Galliet, I., Mounoury, V., Jeulin, D.. Determination of the size of the representative volume element for random composites: statistical and numerical approach. Int J Solids Struct 2003;40:3647–3679.
Ketcham, W.M., Hobbs, P.V.. An experimental determination of the surface energies of ice. Philosophical Magazine 1969;19:1161–1173.
Kipfstuhl, S., Faria, S.H., Azuma, N., Freitag, J., Hamann, I., Kaufmann, P., Miller, H., Weiler, K., Wilhelms, F.. Evidence of dynamic recrystallization in polar firn. J Geophys Res 2009;114:B05204.
Kipfstuhl, S., Hamann, I., Lambrecht, A., Freitag, J., Faria, S.H., Grigoriev, D., Azuma, N.. Microstructure mapping: a new method for imaging deformation induced microstructural features of ice on the grain scale. J Glaciol 2006;52(178):398–406.
Kreher, W.. Residual stresses and stored elastic energy of composites and polycrystals. JMechPhys Solids 1990;38:115–128.
Kröner, E.. Berechnung der elastischen Konstanten des Vielkristalls aus den Konstanten des Einkristalls. Z Phys 1958;151:504–518.
Kröner, E.. Self-consistent scheme and graded disorder in polycrystal elasticity. J Phys F: Metal Phys 1978;8:2261–2267.
Ktitarev, D., Gödert, G., Hutter, K.. Cellular automaton model for recrystallization, fabric, and texture development in polar ice. Journal of Geophysical Research (Solid Earth) 2002;107(B8).
Lahellec, N., Michel, J., Moulinec, H., Suquet, P.. Analysis of inhomogeneous materials at large strains using fast fourier transforms. In: Miehe, C., editor. IUTAM Symposium on Computational Mechanics of Solid Materials at Large Strains. Kluwer Ac. Pub. Dordrecht; 2003. p. 247–258.
Lahellec, N., Suquet, P.. Effective behavior of linear viscoelastic composites: a time-integration approach. Int J Solids Struct 2006;44:507–529.
Lahellec, N., Suquet, P.. On the effective behavior of nonlinear inelastic composites: I. Incremental variational principles. J Mech Phys Solids 2007;55:1932–1963.
Laws, N.. Thermostatics of composite materials. J Mech Phys Solids 1973;21(9).
Laws, N., McLaughlin, R.. Self-consistent estimates for the viscoelastic creep compliances of composite materials. Proc R Soc Lond 1978;A353:251–273.
Lebensohn, R.A.. N-site modeling of a 3D viscoplastic polycrystal using Fast Fourier Transform. Acta Materialia 2001;49(14):2723 – 2737.
Lebensohn, R.A., Brenner, R., Castelnau, O., Rollett, A.D.. Orientation image-based micromechanical modelling of subgrain texture evolution in polycrystalline copper. Acta Materialia 2008;56(15):3914 – 3926.
Lebensohn, R.A., Castelnau, O., Brenner, R., Gilormini, P.. Study of the antiplane deformation of linear 2-D polycrystals with different microstructures. International Journal of Solids and Structures 2005;42(20):5441 – 5459.
Lebensohn, R.A., Kanjarla, A.K., Eisenlohr, P.. An elasto-viscoplastic formulation based on fast Fourier transforms for the prediction of micromechanical fields in polycrystalline materials. International Journal of Plasticity 2012;32â“33(0):59 – 69.
Lebensohn, R.A., Liu, Y., Ponte-Castañeda, P.. Macroscopic properties and field fluctuations in model power-law polycrystals: full-field solutions versus self-consistent estimates. Proc Royal Soc Lond A 2004a;460(2045):1381–1405.
Lebensohn, R.A., Liu, Y., Ponte-Castañeda, P.. On the accuracy of the self-consistent approximation for polycrystals: comparison with full-field numerical simulations. Acta Materialia 2004b;52(18):5347–5361.
Lebensohn, R.A., Montagnat, M., Mansuy, P., Duval, P., Meyssonnier, J., Philip, A.. Modeling viscoplastic behavior and heterogeneous intracrystalline deformation of columnar ice polycrystals. Acta Materialia 2009;57(5):1405 – 1415.
Lebensohn, R.A., Tomé, C.N.. A self-consistent viscoplastic model: prediction of rolling textures of anisotropic polycrystals. Mat Sci and Engin, A 1993;175:71–82.
Lebensohn, R.A., Tomé, C.N., Ponte-Castañeda, P.. Self-consistent modelling of the mechanical behaviour of viscoplastic polycrystals incorporating intragranular field fluctuations. Phil Mag 2007;87(28):4287–4322.
Letouzé, N., Brenner, R., Castelnau, O., Béchade, J.L., Mathon, M.H.. Residual strain distribution in Zircaloy-4 measured by neutron diffraction and estimated by homogenization techniques. Scripta Mat 2002;47:595–599.
Lipenkov, V.Y., Barkov, N.I., Duval, P., Pimienta, P.. Crystalline texture of the 2083m ice core at Vostok Station, Antarctica. J Glaciol 1989;35(121):392–398.
Lipenkov, V.Y., Salamatin, A.N., Duval, P.. Bubbly-ice densification in ice sheets: II. Applications. J Glaciol 1997;43(145):397–407.
Liu, Y., Gilormini, P., Ponte Castañeda, P.. Variational self-consistent estimates for texture evolution in viscoplastic polycrystals. Acta Mater 2003;51:5425–5437.
Liu, Y., Ponte Castañeda, P.. Second-order theory for the effective behavior and field fluctuations in viscoplastic polycrystals. Journal of the Mechanics

and Physics of Solids 2004;52(2):467 – 495.

Lliboutry, L., Duval, P.. Various isotropic and anisotropic ices found in glacier and polar ice caps and their corresponding rheologies. Annales Geophysicae 1985;3(2):207–224.

Llorens, M.G., Bons, P.D., Griera, A., Gomez-Rivas, E., Evans, L.A.. Single layer folding in simple shear. Journal of Structural Geology 2012;(0):–.

Logé, R., Bernacki, M., Resk, H., Delannay, L., Digonnet, H., Chastel, Y., Coupez, T.. Linking plastic deformation to recrystallization in metals using digital microstructures. Philosophical Magazine 2008;88:3691–3712.

Long, M., Becker, T.. Mantle dynamics and seismic anisotropy. Earth and Planetary Science Letters 2010;297:341–354.

Ma, Y., Gagliardini, O., Ritz, C., Gillet-Chaulet, F., Durand, G., Montagnat, M.. Enhancement factors for grounded ice and ice shelves inferred from an anisotropic ice-flow model. Journal of Glaciology 2010;56(199):805–812.

Mandel, J.. Mécanique des milieux continus. Paris, France: Gauthier-Villars, 1966.

Mangeney, A., Califano, F., Castelnau, O.. Isothermal flow of an anisotropic ice sheet in the vicinity of an ice divide. J Geophys Res 1996;101(12):28,189–28,204.

Mangeney, A., Califano, F., Hutter, K.. A numerical study of anisotropic, low Reynolds number, free surface flow of ice sheet modeling. J Geophys Res 1997;102(B10):22,749–22,764.

Mansuy, P., Meyssonnier, J., Philip, A.. Localization of deformation in polycrystalline ice: experiments and numerical simulations with a simple grain model. Computational Materials Science 2002;25(1-2):142–150.

Mansuy, P., Philip, A., Meyssonnier, J.. Identification of strain heterogeneities arising during deformation of ice. Ann Glaciol 2000;30:121–126.

Martín, C., Gudmundsson, G.H., Pritchard, H.D., Gagliardini, O.. On the effects of anisotropic rheology on ice flow, internal structure, and the age-depth relationship at ice divides. J Geophys Res 2009;114:F04001.

Masson, R., Bornert, M., Suquet, P., Zaoui, A.. An affine formulation for the prediction of the effective properties of nonlinear composites and polycrystals. Journal of the Mechanics and Physics of Solids 2000;48(6-7):1203 – 1227.

Masson, R., Zaoui, A.. Self-consistent estimates for the rate-dependent elastoplastic behaviour of polycrystalline materials. J Mech Phys Solids 1999;47:1543–1568.

Mathiesen, J., Ferkinghoff-Borg, J., Jensen, M., Levinsen, M., Olesen, P., Dahl-Jensen, D., Svensson, A.. Dynamics of crystal formation in the Greenland NorthGRIP ice cores. Journal of Glaciology 2004;50:325–328.

Mellor, M., Testa, R.. Creep of ice under low stress. Journal of Glaciology 1969;8:147–152.

Michel, J.C., Moulinec, H., Suquet, P.. Effective properties of composite materials with periodic microstructure: a computational approach. Comp Meth Appl Mech Engng 1999;**172**:109–143.

Michel, J.C., Moulinec, H., Suquet, P.. A computational method based on augmented lagrangians and fast Fourier transforms for composites with high contrast. Comput Model Eng Sci 2000;1:79–88.

Michel, J.C., Moulinec, H., Suquet, P.. A computational scheme for linear and non-linear composites with arbitrary phase contrast. International Journal for Numerical Methods in Engineering 2001;52(1-2).

Miguel, M.C., Vespignani, A., Zapperi, S., Weiss, J., Grasso, J.R.. Intermittent dislocation flow in viscoplastic deformation. Nature 2001;410:667–671.

Miyamoto, A.. Mechanical properties and crystal textures of Greenland deep ice cores. Ph.D. thesis; Hokkaido University; Sapporo; 1999.

Molinari, A., Canova, G., Ahzi, S.. A self-consistent approach of the large deformation polycrystal viscoplasticity. Acta Metall 1987;35:2983–2994.

Montagnat, M., Blackford, J.R., Piazolo, S., Arnaud, L., Lebensohn, R.A.. Measurements and full-field predictions of deformation heterogeneities in ice. Earth and Planetary Science Letters 2011;305(1-2):153 – 160.

Montagnat, M., Buiron, D., Arnaud, L., Broquet, A., Schlitz, P., Jacob, R., Kipfstuhl, S.. Measurements and numerical simulation of fabric evolution along the Talos Dome ice core, Antarctica. Earth and Planetary Science Letters 2012;357-358(0):168 – 178.

Montagnat, M., Durand, G., Duval, P.. Recrystallization processes in granular ice. Supp Issue Low Temperature Science 2009;68:81–90.

Montagnat, M., Duval, P.. Rate controlling processes in the creep of polar ice, influence of grain boundary migration associated with recrystallization. Earth Planet Sc Lett 2000;183:179–186.

Montagnat, M., Weiss, J., Chevy, J., Duval, P., Brunjail, H., Bastie, P., Gil Sevillano, J.. The heterogeneous nature of slip in ice single crystals deformed under torsion. Philosophical Magazine 2006;86(27):4259–4270.

Morgan, V.. High-temperature ice creep tests. Cold Reg Sc Tech 1991;19:295–300.

Moulinec, H., Suquet, P.. A numerical method for computing the overall response of nonlinear composites with complex microstructure. Computer Methods in Applied Mechanics and Engineering 1998;157(1-2):69 – 94.

Nebozhyn, M., Gilormini, P., Ponte Castañeda, P.. Variational self-consistent estimates for cubic viscoplastic polycrystals : the effects of grain anisotropy and shape. J Mech Phys Solids 2001;49:313–340.

Nye, J.. Some geometrical relations in dislocated crystals. Acta Materialia 1953;1:153–162.

Pantleon, W.. Resolving the geometrically necessary dislocation content by conventional electron backscattering diffraction. Scripta Materialia 2008;58(11):994 – 997.

Paterson, W.S.B.. The physics of glaciers. Pergamon, Oxford, 1994.

Pettit, E.C., Thorsteinsson, T., Jacobson, P., Waddington, E.D.. The role of crystal fabric in flow near an ice divide. J Glaciol 2007;53(181):277–288.

Pettit, E.C., Waddington, E.D., Harrison, W.D., Thorsteinsson, T., Elsberg, D., Morack, J., Zumberge, M.A.. The crossover stress, anisotropy and the ice flow law at Siple Dome, West Antarctica. Journal of Glaciology 2011;57(201):39–52.

Piazolo, S., Bons, P.D., Jessell, M.W., Evans, L., Passchier, C.W.. Dominance of microstructural processes and their effect on microstructural development: insights from numerical modelling of dynamic recrystallization. Geol Soc, London, Spec Publ 2002;200:149–170.

Piazolo, S., Borthwick, V., Griera, A., Montagnat, M., Jessell, M.W., Lebensohn, R.A., Evans, L.. Substructure dynamics in crystalline materials: New insight from in situ experiments, detailed EBSD analysis of experimental and natural samples and numerical modelling. Materials Science Forum 2012;715-716:502–507.

Piazolo, S., Jessell, M.W., Bons, P.D., Evans, L., Becker, J.K.. Numerical simulations of microstructures using the Elle platform: A modern research and teaching tool. Journal of the Geological Society of India 2010;75:110–127.

Pimienta, P., Duval, P., Lipenkov, V.Y.. Mechanical behaviour of anisotropic polar ice. In: International Association of Hydrological Sciences, Publication 170. Symposium on The Physiscal Basis of Ice Sheet Modelling, Vancouvert; 1987. p. 57–66.

Placidi, L.. Thermodynamically consistent formulation of induced anisotropy in polar ice accounting for grain rotation, grain-size evolution and recrystallization. Ph.D. thesis; Darmstadt University of Technology; Darmstadt; 2004. Available at http://elib.tu-darmstadt.de/diss/000614/.

Placidi, L.. Microstructured continua treated by the theory of mixtures. Ph.D. thesis; University of Rome, La Sapienza; Rome; 2005.

Placidi, L., Faria, S.H., Hutter, K.. On the role of grain growth, recrystallization, and polygonization in a continuum theory for anisotropic ice sheets. Ann Glaciol 2004;39:49–52.

Placidi, L., Greve, R., Seddik, H., Faria, S.H.. Continuum-mechanical, anisotropic flow model, based on an anisotropic flow enhancement factor (CAFFE). Continuum Mech Thermodyn 2010;22(3):221–237.

Placidi, L., Hutter, K.. An anisotropic flow law for incompressible polycrystalline materials. Zeitschrift für Angewandte Mathematik und Physik (ZAMP) 2005;57:160–181. 10.1007/s00033-005-0008-7.

Placidi, L., Hutter, K.. Thermodynamics of polycrystalline materials treated by the theory of mixtures with continuous diversity. Continuum Mechanics and Thermodynamics 2006;17(6):409–451.

Ponte Castañeda, P.. The effective mechanical properties of nonlinear isotropic composites. J Mech Phys Solids 1991;39:45–71.

Ponte-Castañeda, P., Suquet, P.. Nonlinear composites. Advance in Applied Mechanics 1998;34:171–302.

Ponte Castañeda, P.. Exact second-order estimates for the effective mechanical properties of nonlinear composite materials. J Mech Phys Solids 1996;44:827–862.

Ponte Castañeda, P.. Second-order homogenization estimates for nonlinear composites incorporating field fluctuations. I – Theory. J Mech Phys Solids 2002;50:737–757.

Raabe, D., Becker, R.C.. Coupling of a crystal plasticity finite-element model with a probabilistic cellular automaton for simulating primary static recrystallization in aluminium. Modelling and Simulation in Materials Science and Engineering 2000;8(4):445.

Raabe, D., Roters, F.. Using texture components in crystalplasticity finite

element simulations. Int J Plasticity 2004;20:339–361.

Ricaud, J.M., Masson, R.. Effective properties of linear viscoelastic heterogeneous media: Internal variables formulation and extension to ageing behaviours. Int J Solids Struct 2009;46:1599–1606.

Roessiger, J., Bons, P.D., Griera, A., Jessell, M.W., Evans, L., Montagnat, M., Kipfstuhl, S., Faria, S.H., Weikusat, I.. Competition between grain growth and grain size reduction in polar ice. Journal of Glaciology 2011;57:942–948.

Rolland du Roscoat, S., King, A., Philip, A., Reischig, P., Ludwig, W., Flin, F., Meyssonnier, J.. Analysis of snow microstructure by means of x-ray diffraction contrast tomography. Advances Enginering Materials 2011;13:128–135.

Russell-Head, D.S., Wilson, C.J.L.. Automated fabric analyser system for quartz and ice. J Glaciol 2001;24(90):117–130.

Sanchez-Hubert, J., Sanchez-Palencia, E.. Sur certains problèmes physiques d'homogénéisation donnant lieu à des phénomènes de relaxation. Comptes Rendus Acad Sc Paris 1978;A286:903–906.

Sauter, F., Leclercq, S.. Modeling of the non-monotonous viscoplastic behavior of uranium dioxide. J Nucl Mater 2003;322:1–14.

Schmatz, J.. Grain boundary – fluid inclusion interaction in rocks and analogues. Ph.D. thesis; RWTH-Aachen, Germany; 2010.

Schulson, E.M., Duval, P.. Creep and Fracture of Ice. Cambridge University Press, 2009.

Seddik, H., Greeve, R., Placidi, L., Hamann, I., Gagliardini, O.. Application of a continuum-mechanical model for the flow of anisotropic polar ice to the EDML core, Antarctica. Journal of Glaciology 2008;54(187):631–642.

Seddik, H., Greve, R., Zwinger, T., Gillet-Chaulet, F., Gagliardini, O.. Simulations of the Greenland ice sheet 100 years into the future with the full Stokes model Elmer/Ice. J Glaciol 2012;58(209):427–440.

Seddik, H., Greve, R., Zwinger, T., Placidi, L.. A full stokes ice flow model for the vicinity of Dome Fuji, Antarctica, with induced anisotropy and fabric evolution. The Cryosphere 2011;5(2):495–508.

Solas, D., Gerber, P., Baudin, T., Penelle, R.. Monte Carlo method for simulating grain growth in 3D. influence of lattice site arrangements. Materials Science Forum 2004;467 - 470:1117–1122.

Sotin, C., Tobie, G., J., W.. Europa after Galileo; The University of Arizona Press, Tucson, AZ. p. 85–118.

Suquet, P.. In: Sanchez-Palencia, E., Zaoui, A., editors. Homogenization Techniques for Composite Media. Springer Berlin / Heidelberg; volume 272 of *Lecture Notes in Physics*; 1987. p. 193–198.

Suquet, P., Moulinec, H., Castelnau, O., Montagnat, M., Lahellec, N., Grennerat, F., Duval, P., Brenner, R.. Multi-scale modeling of the mechanical behavior of polycrystalline ice under transient creep. Proceida IUTAM 2011;In press.

Taupin, V., Richeton, T., Chevy, J., Fressengeas, C., Weiss, J., Louchet, F., Miguel, M.. Rearrangement of dislocation structures in the aging of ice single crystals. Acta Materialia 2008;56(7):1555 – 1563.

Taupin, V., Varadhan, S., Chevy, J., Fressengeas, C., Beaudoin, A.J., Montagnat, M., Duval, P.. Effects of size on the dynamics of dislocations in ice single crystals. Phys Rev Lett 2007;99(15):155507.

Taylor, G.. Plastic strain in metals. J Inst Met 1938;62:307–324.

Thorsteinsson, T.. Fabric development with nearest-neighbor interaction and dynamic recrystallization. J Geophys Res 2002;107(B1).

Thorsteinsson, T., Kipfstuhl, J., Miller, H.. Textures and fabrics in the GRIP ice core. J Geophys Res 1997;102(C12):26,583–26,600.

Tommasi, A., Knoll, M., Vauchez, A., Signorelli, J., Thoraval, C., Logé, R.. Structural reactivation in plate tectonics controlled by olivine crystal anisotropy. Nature Geoscience 2009;2(6):423–427.

Urai, J., Means, W., Lister, G.. Dynamic recrystallization of minerals. In: Mineral and Rock Deformation: Laboratory Studies. Hobbs, B.E. and Heard, H.C.; Geophysical Monograph; 1986. p. 161–200.

Vacher, P., Dumoulin, S., Morestin, F., Mguil-Touchal, S.. Bidimensional strain measurement using digital images. Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science 1999;213(8):811–817.

Van der Veen, C.J., Whillans, I.M.. Flow law for glacier ice: comparison of numerical predictions and field measurements. Journal of Glaciology 1990;36:324–339.

Van der Veen, C.J., Whillans, I.M.. Development of fabric in ice. Cold Reg Sci Technol 1994;22(2):171–195.

Varadhan, S., Beaudoin, A., Fressengeas, C.. Coupling the dynamic of statistically distrbuted and excess dislocations. Proc of Science 2006;SMPRI2005, 004:1–11.

Verdier, M., Fivel, M., Groma, I.. Mesoscopic scale simulation of dislocation dynamics in fcc metals: principles and applications. Modelling Simul Mat Sci Eng 1998;6:755–770.

Vu, Q.H., Brenner, R., Castelnau, O., Moulinec, H., Suquet, P.. A self-consistent estimate for linear viscoelastic polycrystals with internal variables inferred from the collocation method. Modelling and Simulation in Materials Science and Engineering 2012;20(2):024003.

Weertman, J.. Creep of ice. In: Whalley, E., Jones, S., Gold, L., editors. Physics and Chemistrey of Ice. Roy. Soc. Canada, Ottawa; 1973. p. 320–337.

Weikusat, I., Kipfstuhl, S., Faria, S.H., Azuma, N., Miyamoto, A.. Subgrain boundaries and related microstructural features in EDML (Antarctica) deep ice cores. J Glaciol 2009;55(191):461–472.

Weiss, J., Grasso, J.R.. Acoustic emission in single crystals of ice. Journal of Physical Chemistry B 1997;101(32):6113–6117.

Weiss, J., Grasso, J.R., Miguel, M.C., Vespignani, A., Zapperi, S.. Complexity in dislocation dynamics: experiments. Materials Science and Engineering: A 2001;309– 310(0):360 – 364. Dislocations 2000: An International Conference on the Fundamentals of Plastic Deformation.

Weiss, J., Marsan, D.. Three-dimensional mapping of dislocation avalanches: Clustering and space/time coupling. Science 2003;299(5603):89–92.

Weiss, J., Miguel, M.C.. Dislocation avalanche correlations. Materials Science and Engineering A 2004;387-389:292 – 296. 13th International Conference on the Strength of Materials.

Weiss, J., Montagnat, M.. Long-range spatial correlations and scaling in dislocation and slip patterns. Philosophical Magazine 2007;87(8-9):1161–1174.

Wenk, H.R., Canova, G., Bréchet, Y., Flandin, L.. A deformation-based model for recrystallization of anisotropic materials. Acta Mater 1997;45(8):3283–3296.

Willis, J.R.. Variational and related methods for the overall properties of composites. Adv Appl Mech 1981;21:2–78.

Wilson, C., Burg, J., Mitchell, J.. The origin of kinks in polycrystalline ice. Tectonophysics 1986;127:27–48.

Wilson, C., Zhang, Y.. Comparison between experiment and computer modelling of plane strain simple shear ice deformation. J Glaciol 1994;40(134):46–55.

Zhang, Y., Jenkins, J.T.. The evolution of the anisotropy of a polycrystalline aggregate. J Mech Phys Solids 1993;41:1213–1243.

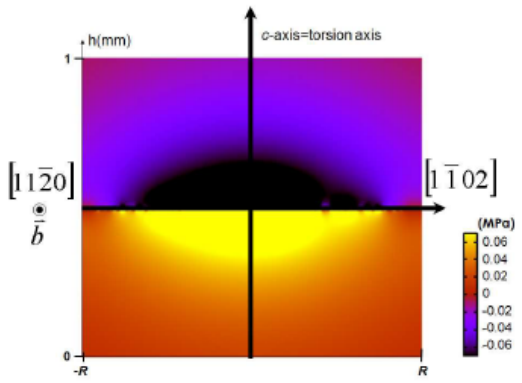


Figure 1: Map of the resolved shear stress in the prismatic system for a torsion boundary made of basal screw dislocations. The cylinder diameter is 1 mm, and the maximum applied stress is 0.1 MPa. From (Chevy et al., 2012)

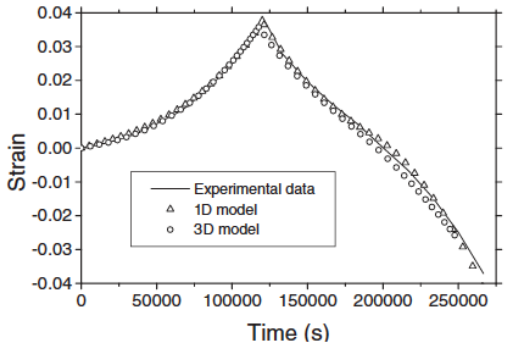


Figure 2: Creep curves in forward and reverse torsion from experiments on single crystals, obtained by 1D and 3D FDM models. From (Taupin et al., 2007)

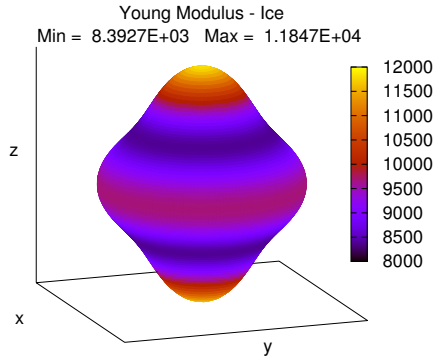


Figure 3: Young's modulus in [MPa] of an ice single crystal with its c-axis aligned with z, at $-16°$C.

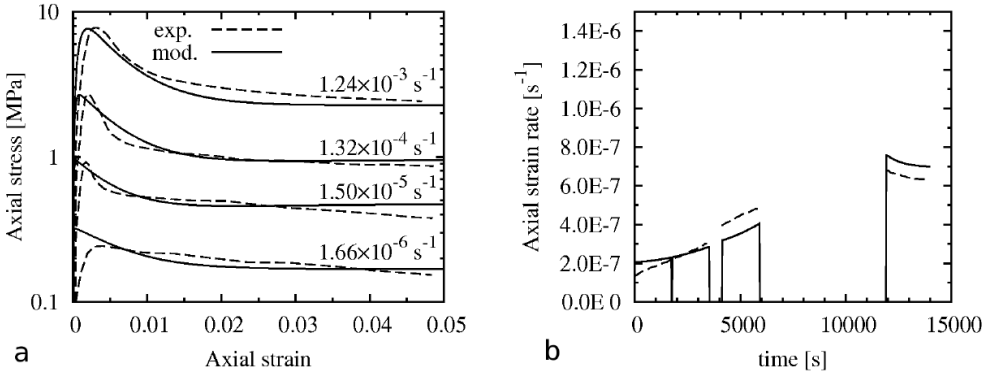


Figure 4: Behavior of an ice single crystal deformed by basal slip. a) Predictions of the model, based on Eq. (6) and with parameters given in Table 1, in comparison with the experimental data of Weertman (1973). Axial strain rates are indicated. b) Prediction of the model in comparison with the results of the recovery tests of Taupin et al. (2008). Temperature is $-10°C$. From (Suquet et al., 2011)

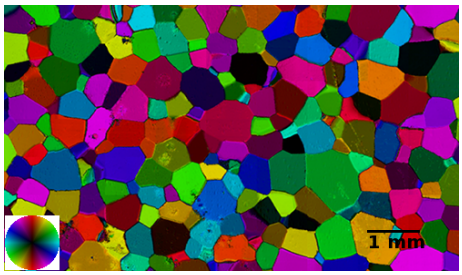


Figure 5: Typical 2D microstructure of an ice polycrystal grown in the laboratory. The color wheel gives the color-code for the c-axis orientation.

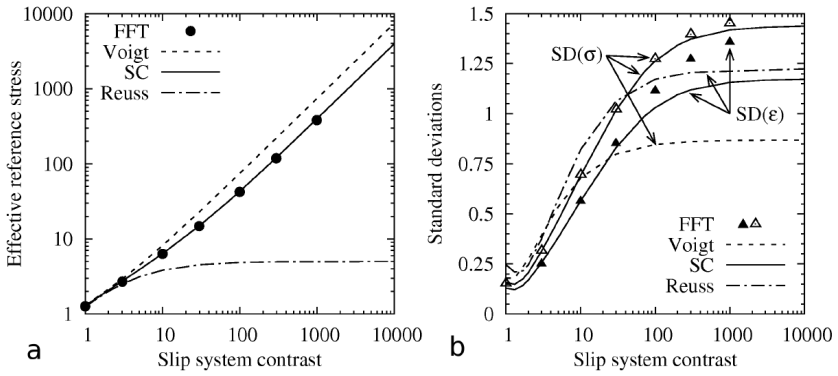


Figure 6: Full-field vs. mean-field behavior for ice polycrystals with random fabric, for a linear viscous behavior ($n = 1$) and various viscoplastic anisotropy (or slip system contrats) at the grains level. a) Effective flow stress $\tilde{\sigma}_0$. b) Standard deviation of equivalent stress and strain rate, normalized by $\bar{\sigma}_{eq}$ and $\bar{\dot{\varepsilon}}_{eq}$, respectively, characterizing field heterogeneities at the polycrystal scale. Results from the linear SC scheme are compared to reference numerical solutions provided by the FFT approach. Reuss and Voigt bounds are also indicated. Note that, for these bounds, standard deviations of stress and strain rate, respectively, do vanish.

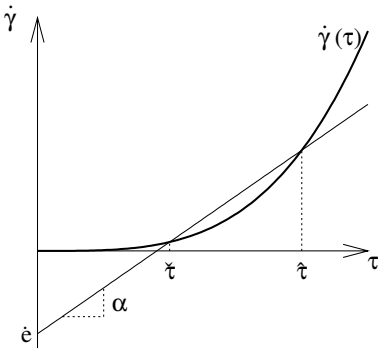


Figure 7: Schematic representation of the linearization between the shear rate ($\dot{\gamma}$) and the stress (τ), to illustrate Eq. (21).

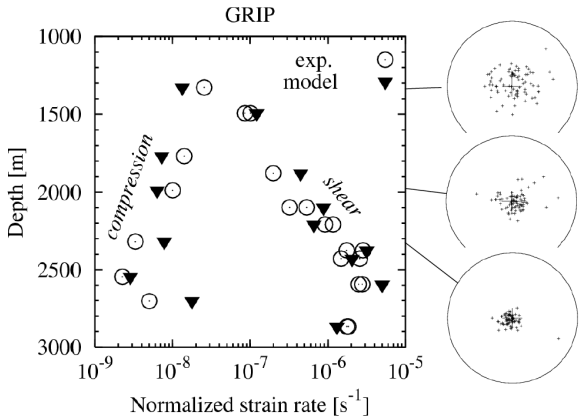


Figure 8: Stationary creep behavior at −10°C calculated by the affine SC model, and compared to experimental data obtained on anisotropic specimens from the GRIP ice core. The *c*-axis pole figures on the right show an increasing concentration of *c*-axes towards the *in situ* vertical direction from the surface of the ice sheet down to ∼ 2600m depth. Experimental data from Castelnau et al. (1998) are expressed for a stress of 1MPa using a stress sensitivity $n = 3$. Points on the left hand side reflect the (hard) behavior under vertical compression, whereas data on the right correspond to (soft) horizontal shear.

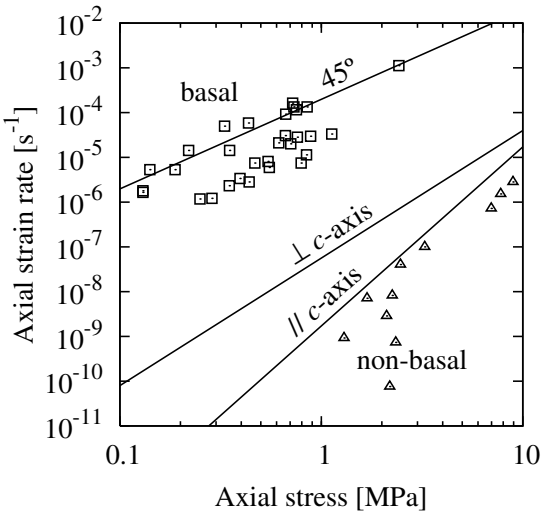


Figure 9: Stationary creep behavior of single crystals at −10°C input in the AFF SC model to get results of Fig. 8 (lines), compared to the data set compiled by Duval et al. (1983) (symbols). Results are indicated for uniaxial compression at 45° from the *c*-axis (activation of basal slip), as well as for compression perpendicular (activation of prismatic slip) and parallel (activation of pyramidal systems) to the *c*-axis. From (Castelnau et al., 2008b)

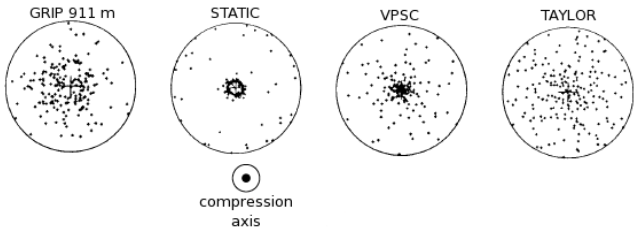


Figure 10: Comparison between fabrics measured along the GRIP ice core (911 m depth), simulated by the static (Reuss), VPSC-tangent, and Taylor (Voigt) approaches.

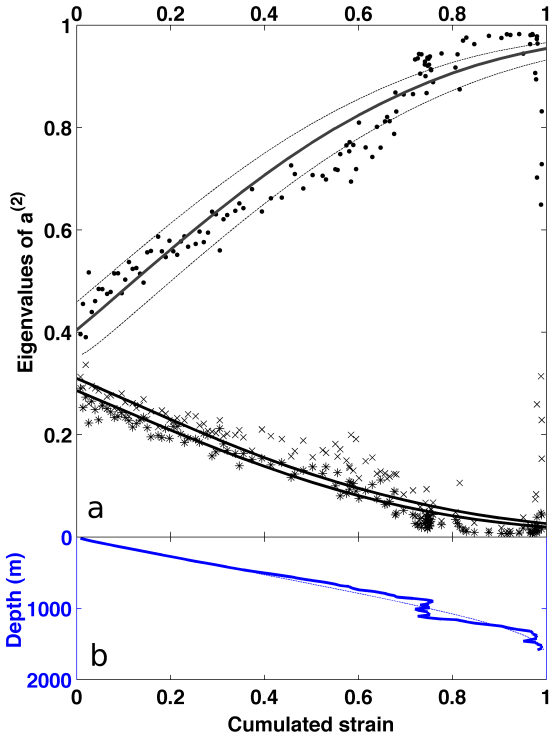


Figure 11: a) Evolution of the eigenvalues of the orientation tensor $\boldsymbol{a}^{(2)} = \boldsymbol{c} \otimes \boldsymbol{c}$ of the fabric along the Talos Dome ice core, as a function of the cumulated compressive strain. Lines = VP-SO model results, dashed line represents the range of fabric evolution modeled with variation of the initial orientation tensor eigenvalue from isotropic (bottom line), as measured at 18 m (central line), more concentrated than measured (top line). Dots, crosses and plus = measurements performed with the Automatic Ice Texture Analyzer (Russell-Head and Wilson, 2001). b) Cumulated in-situ compressive strain as a function of depth as model by the TALDICE-1 chronology (full line) (Buiron et al., 2011).

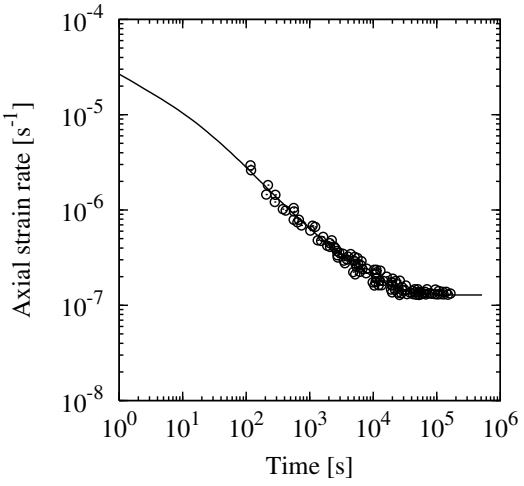


Figure 12: Transient creep response of isotropic ice under an uniaxial compressive stress of 1 MPa predicted with the affine elasto-viscoplastic extension of the self-consistent scheme (line). Model results are compared to the data of Ashby and Duval (1985), expressed for the same loading conditions (points). Strain hardening of prismatic and pyramidal slip systems is taken into account. From (Castelnau et al., 2008b)

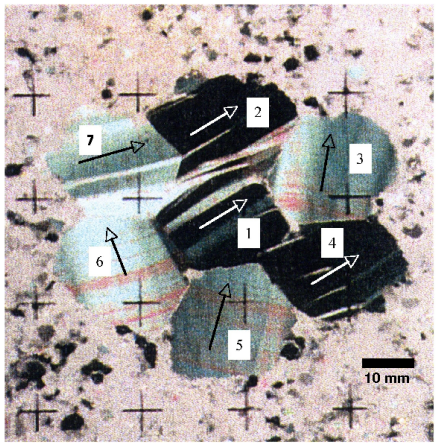


Figure 13: Photograph of a compression creep specimen (after (Mansuy et al., 2000)) between crossed polarizers, after a deformation of 6.6×10^{-2} at -10°C. The corresponding strain rate was 6.0×10^{-8} s^{-1}. The compression direction in vertical in the plane of the photograph.The mean size of each hexagonal grain was 20 mm. Black and white arrows indicate the initial c-axis orientations. Kink bands appear as abrupt changes in color parallel to the c-axis, shear bands are perpendicular to the c-axis direction.

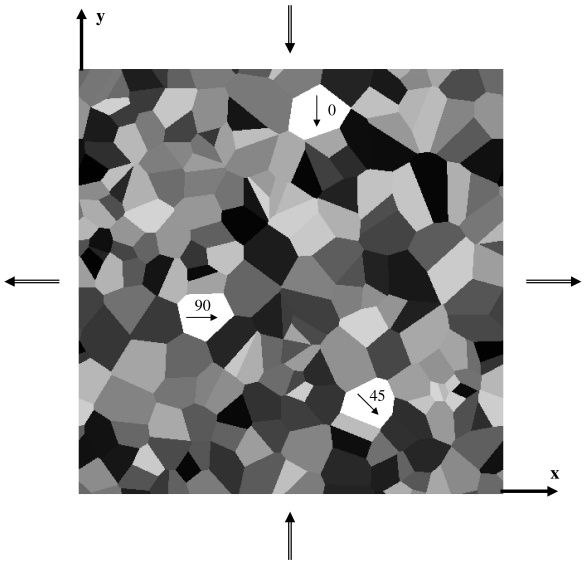


Figure 14: Unit cell containing the cross-sections of 200 columnar grains generated by Voronoi tesellation. The three hand-picked orientations: (0°, 90°, 0°) , (45°, 90°, 0°) and (90°, 90°, 0°), and the extension and shorting directions are also indicated.

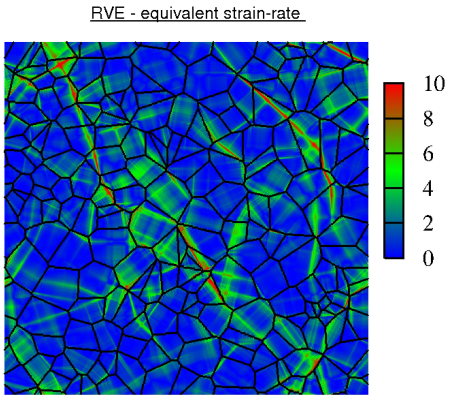


Figure 15: Predicted equivalent strain-rate field over the entire unit cell of Fig. 14, normalized with respect to the average equivalent strain rate ($\dot{\bar{\varepsilon}}_{eq} = 1.15 \times 10^{-8}\mathrm{s}^{-1}$)..

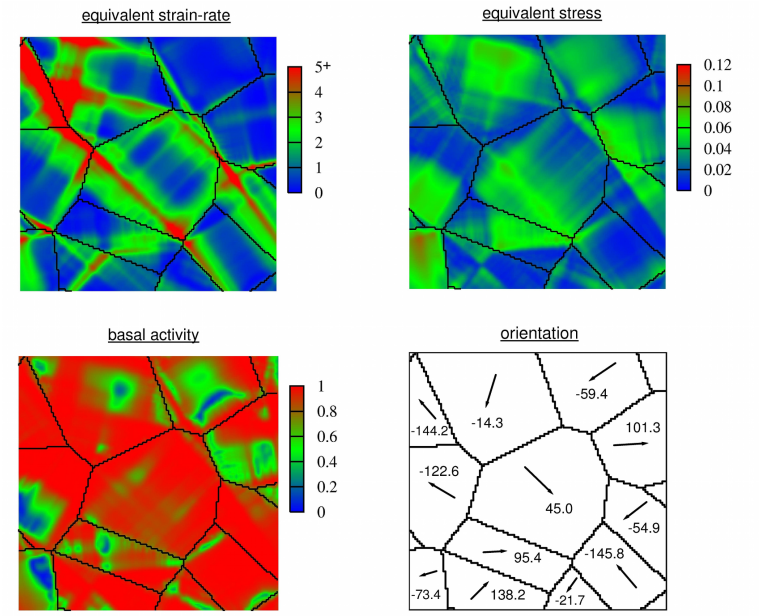


Figure 16: Predicted fields of equivalent strain rate (normalized to $\dot{\bar{\varepsilon}}_{eq}$), equivalent stress (in units of τ^{bas}), relative basal activity, and map of neighbor orientations, for the 45 deg grain and its surroundings.

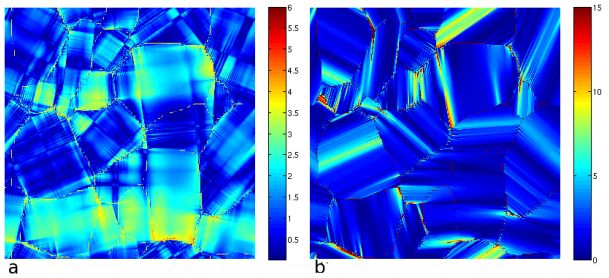


Figure 17: a) Predicted equivalent stress field (in units of τ^{bas}), and b) the misorientation, compared with initial orientation, obtained after 1% strain in a laboratory made microstructure. From (Montagnat et al., 2011)

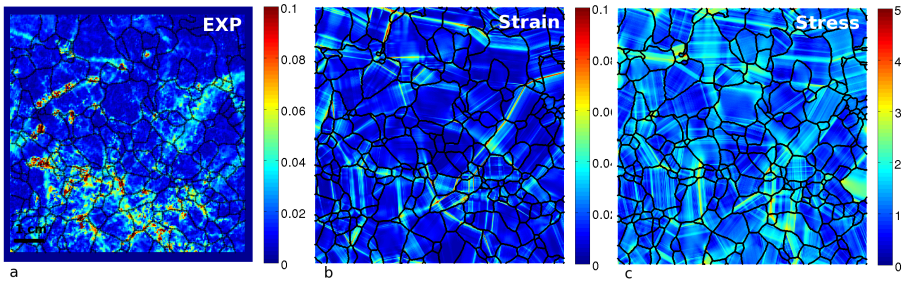


Figure 18: a) Strain field measured experimentally, b) simulated, and c) stress field simulated, after 0.85 % of axial compression. Experimental resolution is about 75×75 pixels, the modeling one is 1024×1024 pixels. From (Grennerat et al., 2012)

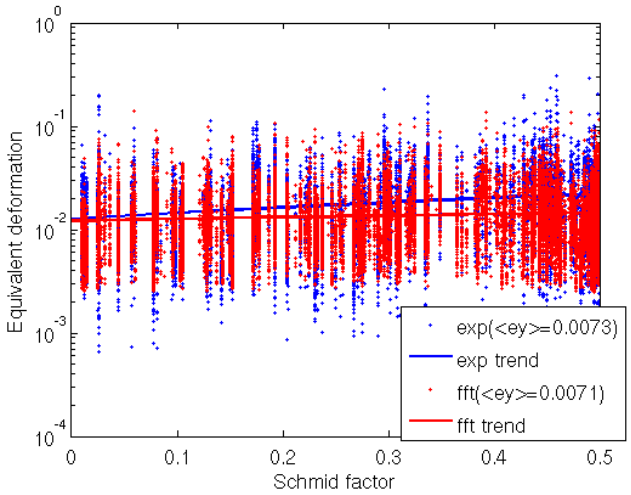


Figure 19: Equivalent strain as a function of the Schmid factor (as a proxy of the orientation). Experimental results are in blue, modeling results in red. Each point is one pixel of the microstructure. The macroscopic strain was 0.7%. From (Grennerat et al., 2012)

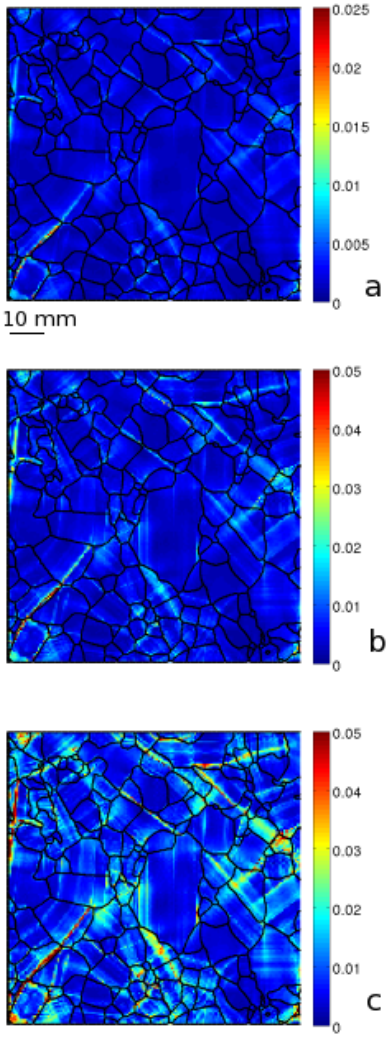


Figure 20: Evolution of the FFT-simulated equivalent strain field during the transient creep of a "2D-1/2" sample of ice, after (a) 0.15%, (b) 0.35% and (c) 0.60% compressive strain (see Grennerat et al. (2012)).

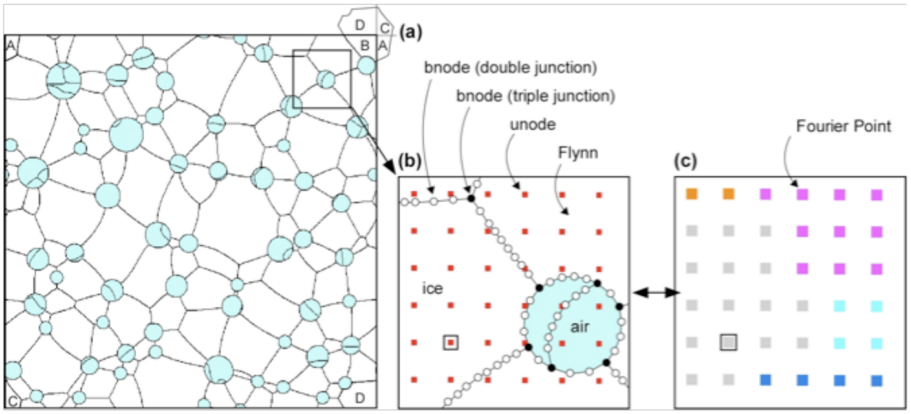


Figure 21: (a) Example of an Elle model: ice (white) with air bubbles (pale blue) (Roessiger et al., this volume). The Elle model has fully wrapping boundaries and grains A to D are in fact one single grain. (b) Close-up showing that grains are defined by flynns (polygons), themselves defined by straight segments that connect boundary nodes (bnodes). A second layer of unconnected nodes (unodes) can be added to keep track of material points. (c) For the FFT module, the microstructure is discretized into a periodic, regular mesh of Fourier Points defined by a characteristic lattice orientation. A direct mapping between unodes layer and Fourier points is established between both codes.

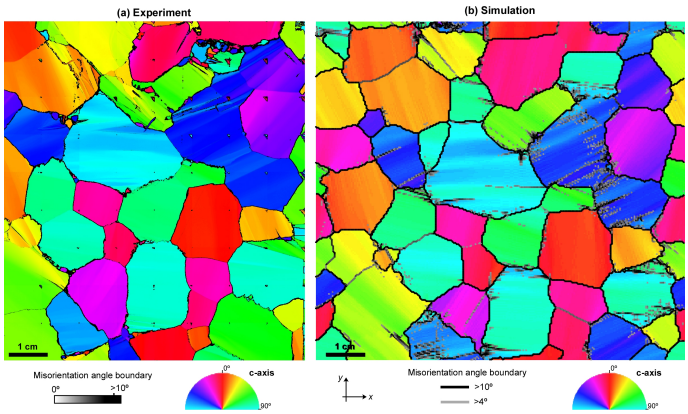


Figure 22: Comparison of (a) physical experiment and (b) numerical simulation after a vertical shortening of 4%. A qualitative equivalence between experiment and simulations is observable, such as correspondence of kink-bands or discontinuous subgrain boundaries at sharp grain boundaries asperities. Colors indicate the orientation of the c-axis respect to the sample reference. Misorientation angle between nodes are indicated in grey (> 4°) and black (> 10°). Triangular patches seen in the experiment are due to erroneous misfit during Automatic Ice Texture Analyzer acquisition.

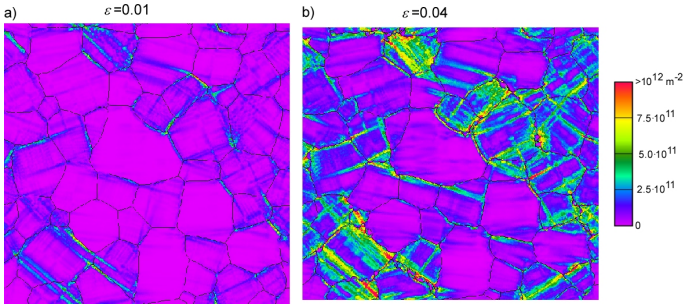


Figure 23: Dislocation density maps after (a) 1% and (b) 4% of shortening. Grain microstructure is indicated by dark lines. Serrated and bulging grain boundaries develop due to grain boundary migration into regions of high dislocation density. New recrystallized grains develop preferentially at triple points and along grain boundaries. Low dislocation densities are typically observed at bulge areas and new grains.

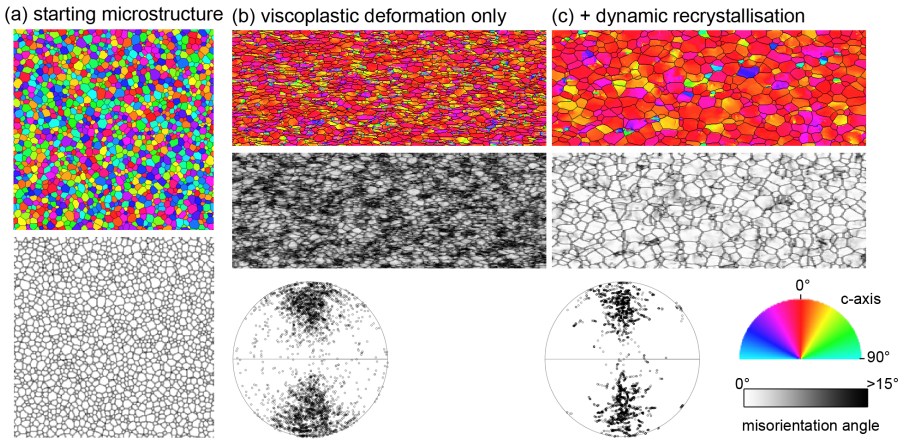


Figure 24: Numerical simulation of polar ice microstructure using the FFT/Elle scheme. (a) Starting microstructure. (b) 40% vertical shortening with only viscoplastic deformation. (c) 40% shortening with viscoplastic deformation coupled with recrystallization. Top row shows c-axis orientations in color and local misorientation in grey. Bottom row shows local misorientation only. C-axis orientation distributions are shown in lower-hemisphere stereoplots.

	τ_{ini}	τ_{sta}	$\dot{\gamma}_0$	n	c	d	e
Basal	0.1	0.022	10^{-6}	2	9	60	0.0003
Prismatic	0.13	1.5	10^{-6}	2.85	9	60	0.0003
Pyramidal	3.875	3.875	10^{-6}	4	9	60	0.0003

Hardening matrix:

	Basal	Prismatic	Pyramidal
Basal	70	125	0
Prismatic	125	110	0
Pyramidal	0	0	0

Table 1: Material parameters used in the full-field simulations for single crystals of ice at $-10°C$. Units are MPa and s^{-1}.

Model	Scale	Type	Applications	Advantages	Limitations
DD	Single crystal (mesoscopic)	Finite Element (FE)	Dislocation dynamics and interactions	Solve physics of plasticity at the dislocation scale - Provide slip system activities and interactions	Computing time-intensive. Small samples. Simplified configurations
FDM	Single to poly-crystal	Full-field (FE)	Plasticity modeling taking into account heterogeneous internal stress field associated with dislocations	Length scaling, makes the link between the complexity of the dislocation field dynamics and the macroscopic behavior - Provide intra-crystalline fields	Computing time-intensive - Limited number of grains
VPSC	Polycrystal	Mean-field	Provides effective visco-plastic behavior of polycrystals, based on a given single-crystal behavior	Texture-induced polycrystal anisotropy, texture evolution for secondary creep, can reach large strains	Very limited information on intra-crystalline fields, elasticity and recrystallization neglected
EVPSC	Polycrystal	Mean-field	Provides effective elasto-visco-plastic behavior at polycrystal level, based on known single-crystal behavior	Captures texture-induced anisotropy during transient creep regime	Very limited information on intra-crystalline fields, limited to small strains

Table 2: Summary of techniques application domains, interest and limitations. DD: Dislocation Dynamics, FDM: Field Dislocation Mechanics, VPSC: Homogenized Polycrystal Visco-Plasticity, EVPSC: Homogenized Polycrystal Elasto-Visco-Plasticity

Model	Scale	Type	Applications	Advantages	Limitations
Mixture with Continuous Diversity (MCD)	Large scale	General continuum theory	General overview of the interactions between microstructure evolution, DRX and ice flow	Effects of microstructure and its evolution via internal variables. Secondary and tertiary creep regimes. Thermodynamically consistent	Mathematically complex. Not implemented numerically yet
CAFFE	Large Scale	Continuum model	Provides effective visco-plastic behavior on the large scale, including fabric development	Captures the effects of fabric development in the secondary and tertiary creep regimes, easy to implement	Stress and strain rate are colinear (scalar effective viscosity). Limited to fabrics represented by a multipole expansion up to fourth order only.
GOLF law	Polycrystal	Phenomen-ological orthotropic non-linear law	Provides orthotropic viscous behavior and fabric development	Efficient, easy-to-use and able to reproduce the response of micro-macro models	Orthotropic fabric restricted to the assumed closure approximation. Non-linear case not validated against micro-macro models yet
Elmer/Ice	Large scale	FE code including GOLF and CAFEE law	Flow of anisotropic polar ice and its fabric evolution	Fabric evolution consistent with the stress field and strain-rate field	Can not be used to simulate the localization of the deformation (diffusion of the fabric induced by interpolation)

Table 4: Summary of techniques application domains, interest and limitations. DRX: dynamic recrystallization

Model	Scale	Type	Applications	Advantages	Limitations
VPSC + DRX	Polycrystal	Mean-field	DRX mechanisms with phenomenological laws, fabric evolution	Fast to run for high strain - easily adaptable to various DRX laws	Too simplified description of DRX mechanisms. No account for intra-crystalline fields
VPFFT	Polycrystal	Full-field	Provides effective visco-plastic behavior and local intra-crystalline fields, 2D and 3D	Microstructural effects on local fields distribution in the secondary creep regime	Microstructure evolution at large strains can only be captured in a crude way. No elasticity and DRX
EVPFFT	Polycrystal	Full-field	Provides effective elasto-visco-plastic behavior and local intra-crystalline fields, 2D and 3D	Microstructural effects on local fields distribution and their evolution during transient creep	Limited to small strains, no microstructure evolution yet, no DRX
VPFFT - Elle	Polycrystal	Full-field	Provides local intra-crystalline fields and microstructure evolution in 2D	Couple local field predictions to DRX mechanisms in the secondary creep regime	No account for local field evolution during transient creep. Limited to 2D (Elle). Rough update of the dislocation field during DRX

Table 3: Summary of techniques application domains, interest and limitations. DRX: dynamic recrystallization, VPFFT: FFT-based formulation for Visco-Plastic polycrystals , EVPFFT: FFT-based formulation for Elasto-Visco-Plastic polycrystals

APPENDIX 4

SIMULATION CODE DESCRIPTION

CONTENTS

1 CODE DESCRIPTION OF WRITTEN ELLE MODULES

The intension of this chapter is to provide people using my code with a detailed description of all the modules I changed and wrote myself. Where useful I included small sections of the original code. The full length of the code is found in the source code files in the appropriate folders however. It is advisable to open that file and browse to the appropriate sections while reading this description. Altogether I completed three major projects and the last of them underwent a major revision after a serious disadvantage of the second approach started to show up. I will start with the description of the second project “a new split code” because this one resulted and is used by the first project “growth + split”. For explanation purposes it is better to understand how the new split works first and why I decided to create it instead of sticking to the old split code.

1.1 NAMING CONVENTIONS

Code	**Meaning**
i	Integer
d	Double

1.2 SPLIT CODE

Generally speaking split is not a process on its own. Although there is the possibility to run the split code directly on an Elle file, the by far most common use is to call the split function from another process. While writing the split code I could think of three different methods which are incorporated in the following functions in the code.

```
i = directionsplit ( i, d, d, d, *i, *i )
Return (int): Error code (to count errors in split for statistical reasons)
Input (int): Flynn number
Input (double): Split direction ( x part of the vector )
Input (double): Split direction ( y part of the vector )
Input (double): Minimum child area (the resulting Flynns have to have at least that size)
Input by reference 2x(*int) : Variables to store and return the Flynn numbers of the resulting child grains

i = randomsplit( i, d, *i, *i )
Return (int): Error code (to count errors in split for statistical reasons)
Input (int): Flynn number
Input (double): Minimum child area (the resulting Flynns have to have at least that size)
Input by reference 2x(*int) : Variables to store and return the Flynn numbers of the resulting child grains

i = directsplit ( i, i, i, *i, *i )
Return (int): Error code (to count errors in split for statistical reasons)
Input (int): Flynn number
Input 2x(int): Start and end node of the Split
Input by reference 2x(*int) : Variables to store and return the Flynn numbers of the resulting child grains
```

The first two are used most of the time because they include topology checks. The difference between both is merely that “randomsplit” determines a random direction for the split before it calls “directionsplit” itself.

The last function “directsplit” doesn’t execute topology checks before the split is carried out. It might only be used for testing purposes or if the process which calls the “directsplit” function already did topology checks itself. However in this chase a custom implementation of the basecode functions is advisable.

Since “randomsplit” is basically the same as “directionsplit” and “directsplit” just calls several basecode functions I will now focus on the description of “directionsplit”

Before the compilation of the code there are two constants set as DEFINES at the top of the file which can be changed. The first one MINDNODES is set to 2 by default. It defines how many double nodes a Flynn must have in order to split. Depending on the switch distance of the file this influences the minimal size of Flynns available for a split. The code won’t insert double nodes in order to split a Flynn and it won’t change triple nodes in any way to achieve a split. That means that a Flynn has to have at least two double nodes to be able to split in two. The second constant SECONDTRY is basically a bool constant. It can be set to 0 or 1 whereas 1 is the default setting. This constant influences how split continues if no split is possible with the set minimal size for child grains. 0 means no split will happen, 1 means that another attempt is made with half the set minimal size.

The function itself will determine all nodes along the Flynn and stores them in an array. In the following main part of the function a series of if statements is carried out which all have to be true in order for the split to happen.

```
if (( check = assignstruct ( &id, dir, num_nodes, &possis )) == 1 ) {
  // step 5 use quicksort to sort the struct
  sortstruct ( dev, 0, possis-1 );
    // start from the first to the last entry in the deviation struct
    for ( j=0, i=0; j<possis && i==0; j++ ) {
      start = dev[j].x;
      end = dev[j].y;
      if ((check = nodes2childs( &id, num_nodes, start, end, &child1, &child2, &nchild1, &nchild2 )) == 1 )
        //check min area of child 1
        if (( test_area = areacheck( &child1, nchild1 )) >= min_area )
          //if ok, check min area of child 2
          if (( test_area = areacheck( &child2, nchild2 )) >= min_area )
            //if ok, check intersections of child 1 with split direction
            if ( intersectioncheck( &child1, nchild1 ))
              //if ok, check intersections of child 2
              if ( intersectioncheck( &child2, nchild2 )) {
                flynnsplit2( flynn, start, end, &child1, &child2, &nchild1, &nchild2, &c1, &c2 );
                //printf("Successfully split flynn %d\n", flynn);
                i = 1;
```

By calling function “assignstruct” the orientations of all possible splits between all double nodes of the Flynn and the given, desirable split direction are calculated. To achieve that, the function calculates the orientation of each vector from the first double node along the Flynn boundary to all other double nodes in relation to the split direction after each other. Next it will calculate the orientation of each vector from the second double node to all other double nodes except the first one and so on until the last calculation between the previous to last and the last double node is made. As side effect the number of possibilities is also returned. The resulting array is then sorted by a quicksort algorithm.

Each entry in the array will now have two integers which are the two node numbers of the start and end node of that possible split, and a double which will be the deviation from the desirable split direction starting with the least possible deviation from the split direction to the largest.

The next function call “nodes2childs” will divide all boundary nodes along the original Flynn in two arrays. Assuming the nodes along the original Flynn have the numbers as shown in Fig 1, the first Array will contain nodes 2, 3, 4, 5 and 6. The second array will contain nodes 6, 7, 8, 9, 10, 11, 12, 1 and 2. Also returned are the sizes of both arrays.

With that information the next two checks will determine whether the size of both childs is not smaller

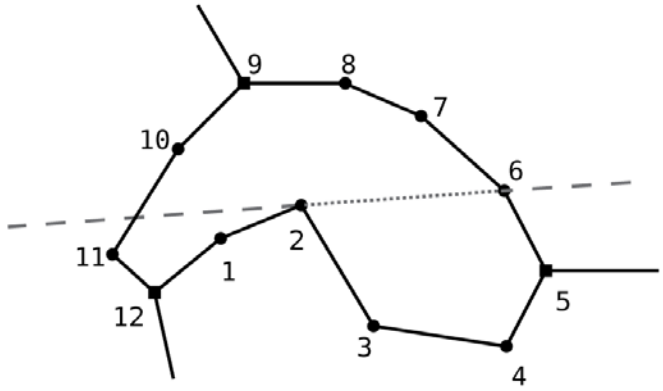


FIG 1: NODE NUMBERING ALONG A FLYNN SHOWING ATTEMPTED SPLIT BETWEEN NODE 2 AND 6. THE DOTTED LINE SHOWS THE ACTUAL SPLIT LINE WHILE THE DASHED LINE SHOWS THE VIRTUAL CONTINUATION OF THAT LINE USED FOR THE TOPOLOGY CHECKS.

than the given minimum child area from the function call. Normally that would be somewhere between 0.4 and 0.48 times the original area of the Flynn.

If both areas are in range the next two checks will determine whether the intended split will cut the original boundary of the Flynn in any location. For standard shaped Flynns that is not a problem in most of the cases. However with certain processes and settings some Flynns will start to develop a quite unusual shape. Imagine a banana shaped grain and the split currently under investigation should

take place between both ends of the banana. If carried out like this it will cut through areas which are not even part of the original Flynn. That would result in corrupt data arrays leading to an unstable simulation and ultimately to the crash of the program.

The topology checks consist of two parts. The first part will put the position of each node of the child in relation to the split direction. The first neighbour node will determine the side of the virtual line representing the split and its continuation (dashed and dotted lines in Fig 1) on which all other nodes have to be in order to not produce an intersection. If the result is that all nodes are on the same side the second part of the test will not be carried out. In the case that a node is on the other side of the split line the second part of the test will compare the length of the split line (dotted line) and the vector connecting the start of the split and the node under investigation. If the vector is shorter or of equal length than the split line the topology check will return an error. If it is longer no error is returned because the split is not compromised. The child grain is just billowing out in an area which is not affected by the split (Fig 1).

Once both topology checks for the two child Flynns are completed without errors the split is carried out and the loop which cycles through all possible splits is stopped. In the case that no split possibility is found which satisfies all requirements a second attempt is made if the SECONDTRY constant is set. Basically this is the same procedure as already described only the required minimal child area is halved.

At the end of the function one of four codes is returned which can be used to count the number of splits on either the first (1) or the second try (3) and the number of cancelled splits which can be the result of the Flynn being too small (2) or another error (0). This is especially useful to check whether it might be advisable to reduce the switch distance in order to get more double nodes along Flynn boundaries. Also for the statistical analysis of the simulations that information is required.

1.3 GROWTH+SPLIT

Since the split part is now explained let me introduce my first process which was used to simulate experiments in our first publication. Basically it is just a combination of the growth code which was one of the first ELLE processes ever written and the split process. At first I used the split process already included in the ELLE package when I started my PhD. However detailed logging and observation showed that in quite many cases this process doesn't split certain grains when it should. I didn't investigate it further, I think however this is related to the triangulate functions which are used in many processes. Basically if you set a certain split chance and you don't check whether the splits are actually successful you assume a split chance in the statistics which is higher than the actual rate. Since the difference in my first attempts was in the order of 5% I thought about possibilities to change the way Flynns are split.

Again there are several operation modes for this process. All parameters required are substituted by the userdata option in the command line.

```
elle_jr_gg_split –u 1 2 3 4 5 6
```

1. Splitmode (int 1, 2, 3)
   Splitmode 1: all grains have the same split chance
   Splitmode 2: every grain > min_area have the same chance to split.
   Splitmode 3: chance increasing. From < min_area = 0% to > max_area = 100%
2. Splitchance (double between 0 and 1)
3. Random forward (int): does a given number of random calls before the value is actually used. Further randomizes the simulation.
4. Starting step (int): In case of a restart. Influences the file numbering. This way a restart can be done in the same directory without overwriting files.
5. Min_area (double): used by Splitmode 2 and 3
6. Max_area (double): used by Splitmode 3

The process will always write out a log file which contains information about how many errors occurred while splitting.

In general every time step all the boundary nodes are moved to reduce the curvature of the whole system. To achieve that in the case of double nodes, a circle is constructed through the node under investigation and its' two neighbouring nodes. The node is then moved towards the centre of the circle (calculated using a vector) by a small amount. The magnitude of the movement is modified by the

settings for parameters like timestep, switchdistance and speedup. For triple nodes the calculation is basically the same. However three circles are constructed and the resulting vectors are added together.

Once all modes have been moved the split part takes over operation and depending on the setting for split mode either all Flynns or just a portion of Flynns will have a chance to split in two. This is determined by a random number generator which generates double numbers between 0 and just short of 1. If a chance if 25% (0.25) for all grains is set. Every result of a number < 0.25 will lead to a split of the Flynn. After the loop cycled through all Flynns the overall working loop will start the next time step and continue again with the growth loop for all nodes.

1.4 POLY-PHASE GRAIN BOUNDARY MIGRATION

This process is the largest piece of code which has been developed during my PhD. It underwent one major revamp and started out with a quite simple approach which was soon abandoned due to technical issues. In section 1.4.1 I will describe the issues with the simple first approach. Continuing in section 1.4.2 will be a broad description of the first working approach of this process. However with time it got clear that this approach had one major disadvantage which was then overcome by a revision of the code (section 1.4.3) with a different handling of parameters.

1.4.1 TWO-PHASE GROWTH

The first idea on how to realize two phase grain growth was the application of a simple geometrical rule. Imagine a triangle as shown in Fig 2 with three nodes in each corner.

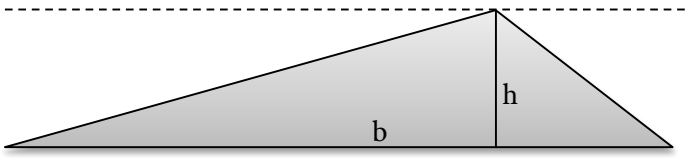


FIG 2 SHOWS A TRIANGLE WITH A NODE IN EACH CORNER. IF MOVING THE MIDDLE NODE ALONG A LINE PARALLEL TO THE CONNECTION LINE BETWEEN THE TWO OTHER NODES, THE AREA WON'T CHANGE.

Using the equation $A = \frac{1}{2}bh$ always yields the same area since neither the base nor the height change during a movement of the upper corner along a line parallel to the base line which connects the other two corners. Now that would work as an area conservative mechanism to move double nodes if we restrict their movement to one parallel to a line connecting their two neighbour nodes. For triple nodes we thought allowing a free movement for them wouldn't affect the outcome very much since only a minority of all nodes in the simulation are actually triple nodes. It turned out that restricting the

movement in this way affected microstructure evolution very strong and the results were far from any natural example we had. This basically meant the simple way wouldn't work and we had to come up with a more complex function to keep areas constant.

1.4.2 POLY-PHASE GRAIN BOUNDARY MIGRATION (USING B-NODES)

Basically this approach also started off with a simple idea. I will frame it shortly but I won't describe it in full detail since it is a discontinued part of the code. Some functions remain in the code because they might be reused in further development which is only possibly by using b-nodes. The idea was to give each node an additional area energy for each phase. At the start of the simulation that should be 0. Movement of the nodes during the simulation most of the time results in changes of area of the neighbouring Flynns. This area change was recorded for each node and for each phase neighbouring this node separately. Area loss for a phase resulted in negative values, gain in positive ones. Nodes which only had neighbouring Flynns of the same phase were skipped because the net gain and loss of area for this phase was zero anyway. These values then influenced the movement of the nodes since the area energy also contributed to the calculation of the energy contours during the trial position step. In the equation used it wasn't important whether the values were positive or negative. The larger they were the larger was the area energy. The goal was to keep the nodes close to an area energy of zero. Now leaving the code at this stage would have resulted in moving nodes around their original position but no development of the whole microstructure. This problem was solved by different mechanisms of "diffusion" of this area energy. The simplest mechanism was to distribute the area energy for each phase equally between all nodes neighbouring this phase after each step. We called this "infinite" diffusion because it simulated a very large diffusion coefficient regardless of the material between the nodes. One can probably think of this like diffusion in hot gas. The second mechanism was similar with a major difference in checking the material between the nodes. It only used infinite diffusion between nodes which were actually connected to each other through a boundary. This diffusion was called cluster diffusion. One can probably think of this as simulation of melt pockets in a crystallized structure. Diffusion in the melt is much faster than in the solid grains. The last technique was implemented as Fick's diffusion. Depending on the setting each node only diffused to the immediate neighbouring nodes and so on. Probably this function might be of use later to simulate diffusion of one phase through grain boundaries of another phase.

During continuous simulations with this code it seemed to work fine at first because all simulations I carried out had settings for high wetting angles between the minor and the major phase. That meant that the minor phase formed round areas within the major phase which were not connected to each other. Due to the setup of the data structure this there is not much change in the boundary nodes in an

environment like this. Later, after Philipp, a diploma student, was making use of the code himself to simulate the evolution of melt distribution we discovered that with settings for low wetting angles between the phases the change in the boundary nodes has a major impact. With settings for low wetting angles between the phases the minor phase might start as individual areas in the major phase. However it is going to be distributed along the grain boundaries and forms connections to each other. Because of that in the beginning there might be just a few boundary nodes in contact with the minor phase but because it keeps being distributed more and more boundary nodes come in contact with it. The major flaw of the code in this case was that it doesn't keep track of the amount of boundary nodes. Also if new nodes are inserted the usually start off with no attributes. That means the build-up of "pressure" of the area energy is counteracted because it gets distributed between more and more nodes during diffusion. This resulted in significant area loss of the minor phase in most cases. Several attempts were made to counteract this problem like keeping track of the amount of nodes or to give new nodes always the average value of the neighbouring nodes. In the end there was no satisfactory result and this lead to the third approach using Flynns instead of boundary nodes for area energy calculation.

1.4.3 POLY-PHASE GRAIN BOUNDARY MIGRATION (USING FLYNNS)

This version of the poly phase grain growth code represents the latest development stage. Due to the different steps which in the end lead to this approach it has to be said that there are a couple of functions still in the code which are not used by the current process anymore and are only related to previous developments. The code will be cleaned up in the future. However until now I didn't find time to complete that step. Because the last version of the code had problems with settings for low wetting angles, I thought of a different approach. It is similar to the melt code of J. Becker, however, it keeps the fraction of phase constant for each cluster of Flynns. Clusters consist of one or more Flynns of the same phase which at least share one boundary. Clusters of one Flynn are theoretically not a cluster. To avoid the necessity to distinguish between single Flynns and actual Clusters I treated single Flynns also as a special kind of Cluster. The disadvantage of this approach is that as it is there is no possibility to implement slow diffusion along the phase boundary of a cluster or even along grain boundaries between other phases. This was one of the reasons why I didn't clean the code and get rid of all old functions. Since they worked with the boundary nodes my idea was to implement this ability later by recycling old functions.

1.4.3.1 CONFIG FILE AND SETTINGS

Like the previous version of the code, this process needs the “phase_db.txt” configuration file to be in the same directory as all the other experiment related files. Some of the settings in this file are also outdated and related to previous versions of the code. In the following I will highlight the important sections and mark outdated sections in grey. The grey parts have to stay in the file for now, otherwise the file reading would be messed up. You can write your own comments in the file. They should always start with a #.

```
#########################################################
######             Number of phases.               ######
#########################################################

2

#########################################################
######             Phase Properties                ######
# A) Phase Number
# B) Infinite Diffusion (y/n)
# C) Cluster Diffusion (y/n)
# D) Ficks Diffusion (Number of diff steps per time step)
# E) Exponent for scaling
# F) Constant for scaling
# G) Kappa for Ficks diffusion
# H) Merge (y/n)
#########################################################

0 0 0 0 2 12000 2e-9 0
1 1 1 0 2 12000 2e-9 0

#########################################################
######         Phase Boundary Properties           ######
# Boundaries are defined by A and B
# A) Phase Number one
# B) Phase Number two
# C) Mobility of these boundary segments
#    below -10ÂºC 7.5e-5 (Duval Book - Creep and Fracture of Ice)
#    above -10ÂºC 1.0e-4 (P. Duval and O. Castelnau, Dynamic Recrystallization of Ice in Polar Ice […]
# D) Surface Energy of these segments
# E) GB Activation Energy (Q)
# mobil = mobility * exp( -(Q ) / ( R * T ) );
#########################################################

0 0 0.023 0.065 51.1e3
0 1 0.023 0.032 51.1e3
1 1 0.038 0.0032 51.1e3

#########################################################
######               MELT TRACKING                 ######
# A) Use the Unode layer to track the given phase.
#    ( -2 --> don't do tracking )
#########################################################

-2
#########################################################
######              CLUSTER_TRACKING               ######
#########################################################
# The cluster tracking multiplier energy function is
# defined by these values...
# area_percentage = (area_new – area_old) / (area_old)
# area_multiplier = A * (area_percentage) ^ D
#
# A B C D
#########################################################
0.1 0 0 2
```

The first section highlighted denotes the number of phases in the experiment. Theoretically the code should work for more than two phases. However it has only been extensively tested for two phases. In

the “Phase Properties” section only the first three numbers are important at the moment. The others have to stay there to keep the file reading in line. Basically the first number denotes the phase number. The two following numbers are either 0 for the most common phase and 1 for all other phases. The “Phase boundary properties” section is the most important one since it sets surface energy and boundary mobility for all different boundaries. The first two numbers denote the phases on both sides of the boundary. The following numbers are mobility, surface energy and grain boundary activation energy. The “Melt tracking” section is only used if you want to use unodes to track the evolution of a specific phase. And the last part, the “Cluster tracking” section is used to set the scaling parameters of the area energy function. So far only A and D are used.

There is another set of configuration setting in the beginning of the cpp file which basically tells the code which Elle storage attribute it should use. “iFlynnPhase” is the attribute for the phase number. “iFlynnCluster” is the attribute for the cluster number (which is the original area of the cluster). “iUnodePhase” and “iUnodeConc” are used for phase tracking which is explained later on in the code. However these values are only used if phase tracking in the config file is set to -1. For all larger numbers fixed storage attributes are used (U_ATTRIB_A, B and C). “iUnodeUpdateMethod” also influences the approach on how unode phases are updated during phase tracking. The next three variables called i, d and bCheckEnergy were used during debugging of the code. At the moment all parts using these don’t do much. In general they can be reactivated and used for energy checking. The idea was to get an idea on writing a function which determines the scaling scalar automatically. The last constant “iMinTjs” is used by the “ElleCheckTripleJ” function. If it is larger than 2 only Flynns with more triple nodes than the set number are checked for possible triple node neighbour switches. That has for the moment statistical reasons which are also explained later, where the constant is actually used in the code.

```
int iClusterNodeCount = N_ATTRIB_A; //not used atm
int attrib[2] = { N_ATTRIB_B, N_ATTRIB_C };
int iFlynnPhase = F_ATTRIB_A;
int iFlynnCluster = F_ATTRIB_C;
int iUnodePhase = U_ATTRIB_C;
int iUnodeConc = U_ATTRIB_A;
//
int iUnodeUpdateMethod = 1;
// 1 = Find Unodes in a Flynn and update them accordingly with the FlynnPhase in iUnodePhase.
// 0 = Find according Flynn for each Unode and update them with the FlynnPhase in iUnodePhase.
double dCheckEnergy[4];
int bCheckEnergy = 0;
int iCheckEnergy = 0;
int iMinTjs = 0;
```

In addition to these two configuration settings the code will create different files during runtime. Probably the most important one to understand the code is the “initial_stuff.txt” file. If it is not present when starting the experiment it will be created in the same directory. Depending on the file being there

or not tells the code to either assume a new simulation or a restarted/continued one. On the very first step no values which are used for cluster tracking and energy management have been stored in the attributes of the Flynns. That means if the file is not present. The code will calculate these variables and store them in the corresponding attribute slots overwriting everything which has been stored there before. If the file is present the code assumes that this step has already been done and the experiment is a continuation of an old experiment. If that is not the case and the file has not been deleted by accident although the simulation is new. The code will crash or end up in a close to infinite loop because all values it needs are either invalid or everything is set to 0 or 1. That means you have to make sure to delete the file if you want to restart from the beginning and you have to make sure it is in the same directory when you want to restart from a later step or continue after running some other function on the experiment. The file itself contains the areas the phases occupied in the very first step.

1.4.3.2 START THE MAIN FUNCTION AND THE CLUSTER TRACKING CLASS

The main function of this process (GBMGrowth) will initialise the clusterTracking class in the beginning.

```
// Initialize the clusterTracking class...
clusterTracking clusters;
if ( clusters.writeInitialData("initial_stuff.txt") ) {
  clusters.setClusterAreas();
  clusters.checkDoubleClusterAreaLoop();
  if ( phases.p_track == -1 )
    UnodePhaseUpdate();
}
```

If this is the first run and the Initial_Stuff.txt file is not present it will run additional functions. During initialisation this class will check with arrays that were created during the parsing of the config file whether there are phases in the experiment that will use cluster tracking. If cluster tracking should work both switches for infinite diffusion and cluster diffusion have to be activated because only Flynns which diffuse their area instantly can also use this approach of cluster diffusion.

```
for ( int i = 0; i < phases.no_phases; i++ ) {
  if ( phases.phasep[ i ].infinite_diff == 1 )
    if ( phases.phasep[ i ].cluster_diff == 1 ) {
      lClustDiffPhases.push_back( i );
      lAllPhases.push_back( i );
      vClusterPhases.push_back( i );
    }
    else {
      lInfDiffPhases.push_back( i );
      lAllPhases.push_back( i );
    }
  else {
    lFicksDiffPhases.push_back( i );
    lAllPhases.push_back( i );
  }
}
```

After that, additional parameters which are only important for the scaling of the energy functions during cluster tracking will be parsed from the config file. In the end it will set a constant which is used to shift cluster areas by a small amount in the unusual case that two clusters have exactly the same area. Then the constructor will call the "findClusters" function. At the end an array is created which will contain all clusters of the different phases which should use cluster diffusion. To achieve that it will create two vectors containing all Flynns and their phases respectively.

```
<vector> vFlynn              → Contains the Flynn numbers
<vector> vFlynnPhase         → Contains the phase numbers of the Flynns
```

If both vectors don't have the same size at the end of the loop an error message will be printed. In any other case the cluster determination is started by calling the function "getClusters". This function is at the moment a slightly modified version of the original from the second version. It makes use of two lists and three vectors where one of the vectors is actually a vector containing another vector on each position and one contains that vector again.

```
<vector> vCluster                    → Is used temporary to find Clusters
<vector<vector>> vClusters           → Is used to store all clusters of the current phase
<vector<vector<vector> vPhaseClusters  → Contains the Flynn numbers of all Clusters of all Phases as vectors.
<list> lOriginal                     → All Flynns of the current phase are stored here
<list> lNeighbour                    → Is used as temporary storage of neigbouring Flynns
```

To clarify the setup I've drawn a small sketch of the data structure (Fig 3)

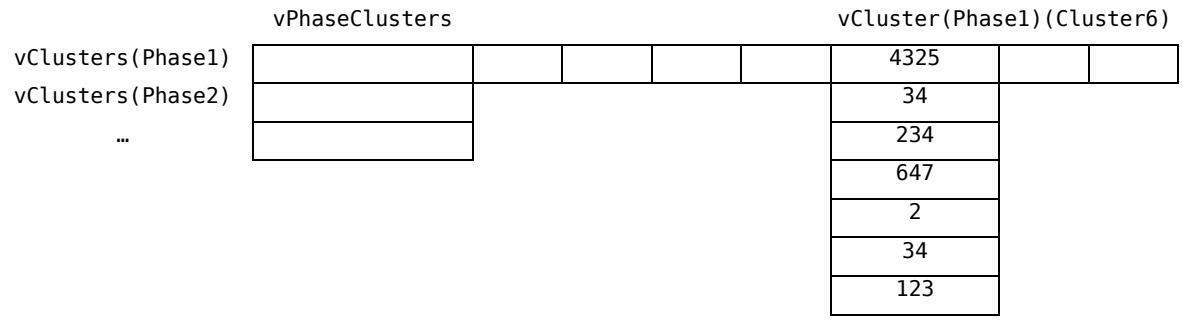


FIG 3 SETUP OF THE CLUSTER DATA STRUCTURE. VPHASECLUSTERS CONTAINS A VECTOR (VCLUSTERS) FOR EACH PHASE THAT USES CLUSTER DIFFUSION. THAT VECTOR CONTAINS OTHER VECTORS (VCLUSTER) FOR EACH CLUSTER OF THAT PHASE. AND THAT VECTOR CONTAINS ALL FLYNN NUMBERS THAT BELONG TO THAT CLUSTER.

The first for loop will cycle through all phases which should diffuse using cluster diffusion. In general that will be a very short vector containing only one or two entries. This way the cluster initialising is done separately for each phase. The next for loop will cycle through all Flynns and check whether their phase is equal to the phase under investigation. If that is the case their Flynn number is pushed to one of the lists (lOriginal). At the end of this loop all Flynns of the current phase should be on that list. To make sure that there are no double entries the list is sorted and all additional, identical entries are

deleted. While there are entries in the lOriginal list, the first entry is transferred to the vClusters vector. The next loop will cycle through this vCluster vector which at the moment only consists of one entry. However usually it will grow if it is not a one Flynn cluster. The loop will find all neighbour Flynns of the current Flynn and puts them on the lNeighbour list. All entries in that list are checked whether they have the same phase as is currently under investigation. If not the Flynn number is deleted from the neighbour list. If they do it will be checked whether the same Flynn number is already in the vCluster vector. If not it will be transferred to the vector, if it is it will just be deleted from the neighbour list. If there was actually one or more neighbours the vCluster loop continues with the next entry. If none was found the vCluster size is still one and the loop therefore finishes. At its end the vCluster vector containing all Flynn numbers of a cluster is pushed to the vClusters vector which in the end will contain all clusters of the current phase as vectors. After the lOriginal list is empty and all entries have been checked that vector is pushed to another vector which in the end will hold information about all clusters of all phases which diffuse by cluster diffusion.

```
while ( lOriginal.size() > 0 ) {
  vCluster.clear();
  vCluster.push_back( lOriginal.front() ); // put the first flynn into the cluster list
  lOriginal.pop_front(); // delete that element from the list

  for ( int n = 0; n < vCluster.size(); n++ ) {
    lNeighbour.clear(); // clear the neigbour list
    ElleFlynnNbRegions( vCluster.at(n), lNeighbour ); //find neighbours for the current flynn (n) in the cluster list

    // check whether any flynn in the neighbour list matches the current phase (i)
    // as long as there are entries in the neighbours list do the following
    while ( lNeighbour.size() > 0 ) {
      ElleGetFlynnRealAttribute( lNeighbour.front(), &temp_double, iFlynnPhase ); // get phase from flynn
      temp_int = (int) temp_double;                          // convert to int
                                                             //compare to current phase
      if ( temp_int == vClusterPhases.at( z ) ) {            // if Flynn has the same phase
                                                             // look whether the Flynn is already on the cluster list
        int iOnList = 0;
        for ( int i = 0; i < vCluster.size() && iOnList == 0; i++ )
          if ( lNeighbour.front() == vCluster.at(i) )
            iOnList = 1;
        if ( iOnList == 0 ) {                                // if NOT
          vCluster.push_back( lNeighbour.front() );          // add flynn to cluster list
          lOriginal.remove( lNeighbour.front() );            // remove that flynn from the original phase list
          lNeighbour.pop_front();                            // remove it from the neighbours list
        }
        else                                                 // if it is
          lNeighbour.pop_front();                            // just remove it from the neighbours list
      }
      else                                                   // if Flynn has not the same phase
        lNeighbour.pop_front();                              // just remove it from the neighbours list
    }
  }
  vClusters.push_back(vCluster); //vector which contains all the flynns beloning to a cluster is put in another vector
}
vPhasesClusters.push_back( vClusters );
```

After this triple vector is created the function “getClusterAreas” will calculate the areas of each cluster.

```
for ( int z = 0; z < vPhasesClusters.size(); z++ ) {
  vClusterArea.clear();
  for ( int i = 0; i < vPhasesClusters[z].size(); i++ ) {
    dClusterArea = 0.0;
    for ( int j = 0; j < vPhasesClusters[z][i].size(); j++ ) {
      dClusterArea += ElleRegionArea( vPhasesClusters[z][i][j] );
    }
    vClusterArea.push_back( dClusterArea );
  }
  vPhasesClusterAreas.push_back( vClusterArea );
```

If the "initial_stuff.txt" file is not present the just calculated areas are written into the set Elle Flynn attribute, overwriting everything that has been there. That means that if the file is deleted during the experiment all areas are basically reset. It will also prevent two clusters from having exactly the same area, which would prevent the cluster tracking system from working properly, by calling "checkDoubleClusterArea".

1.4.3.3 THE MAIN LOOP

After the clusterTracking class has been constructed everything is set to enter the main loop which cycles through all nodes for each time step. The nodes are shuffled each time step to prevent the same node from moving at exactly the same time in a time step. For each node there are four trial positions which are equally displaced from the nodes original position in both directions horizontally and vertically. The energy at each trial position equals the length of each segment, which is the distance between the node and one of the neighbouring nodes, times the set energy for that phase boundary (Eq. 1). Of course the third segment is only significant if the node is a triple node.

$$E = \left((l1 * e1) + (l2 * e2)[+(l3 * e3)]\right) \qquad (1)$$

With the exception that the energy is read from the config file rather than somewhere else, this code hasn't been changed since it has been originally developed by J. Becker, et al. [(1)] To keep the areas of the phases more or less constant another form of energy was introduced which counteracts the natural reduction of energy once the area change of a cluster compared to its original area is larger than 0. This idea is explained in detail in J. Roessiger, et al. [(2)] In code form the function calls a member of the clusterTracking class called "returnClusterAreaEnergy". This function will determine all neighbouring Flynns of the node.

```
for ( int i = 0; i < vClusterPhases.size(); i++ ) {
  vClusterPhaseFlynns.clear();
  for ( int j = 0; j < iNodeType; j++ ) {
    ElleGetFlynnRealAttribute( iFlynns[j], &dFlynnPhaseCheck, iFlynnPhase );
    iFlynnPhaseCheck = (int) dFlynnPhaseCheck;
    if ( iFlynnPhaseCheck == vClusterPhases[ i ] ) {
      vClusterPhaseFlynns.push_back( iFlynns[j] );
    }
  }
  if ( vClusterPhaseFlynns.size() > 0 )
    vPhaseClusterFlynns.push_back( vClusterPhaseFlynns );
```

```
}

if ( vPhaseClusterFlynns.size() == 0 )
  return 0.0;

double dClusterAreaEnergy = 0;
vector<double> vClusterAreaEnergy = clusterTracking::returnClusterAreaChange ( vPhaseClusterFlynns, iNode, xyLoc );

for ( int i = 0; i < vClusterAreaEnergy.size(); i++ )
  dClusterAreaEnergy += vClusterAreaEnergy.at ( i );

return dClusterAreaEnergy;
```

At the end of the loop the Flynns are put into a vector called vPhaseClusterFlynns in regard of the phase they belong to. That means calculation only continues if there is at least one neighbouring Flynn which belongs to a phase that uses cluster diffusion. If not this term is 0 and calculation stops. If there are entries in the vector the function “returnClusterAreaChange” is called to return the actual area change of all clusters neighbouring the node. Each entry in vPhaseClusterFlynns consists of another vector which contains all neighbouring Flynn numbers which belong to the same phase. Let me give an example which might make the setup clearer. Imagine a triple node. That means there are three neighbouring Flynns. Two of them belong to the same phase, one to another. All of them are set to cluster diffusion. That means vPhaseClusterFlynns contains two vectors. One only carries one Flynn number and the other contains two Flynn numbers. The “returnClusterAreaChange” function now calculates the area change for all Flynns.

```
for ( int i = 0; i < vPhaseFlynns.size(); i++ ) {
  vPhaseClusterAreaChange.push_back ( 0 );
  ElleGetFlynnRealAttribute( vPhaseFlynns[i][0], &dClusterArea, iFlynnCluster );
  for ( int j = 0; j < vPhaseFlynns[i].size(); j++ ) {
    vPhaseClusterAreaChange.at ( i ) +=  returnFlynnAreaChange ( vPhaseFlynns[i][j], iNode, xyLoc );
    if ( j > 0 ) {
      ElleGetFlynnRealAttribute( vPhaseFlynns[i][j], &dClusterAreaCheck, iFlynnCluster );
      if ( dClusterArea != dClusterAreaCheck ) {
        cout << "WARNING: Stored Clusterareas in the Flynns […] are not the same!!" << endl;
        clusterTracking::findClusters();
        clusterTracking::findSplit();
        clusterTracking::findMerge();
        clusterTracking::checkDoubleClusterAreaLoop();
        ElleGetFlynnRealAttribute( vPhaseFlynns[i][0], &dClusterArea, iFlynnCluster );
      }
    }
  }

  vClusterArea.push_back( dClusterArea );


  // get the current area
  for ( int z = 0, bFound = false; z < vPhasesClusters.size() && bFound == false; z++ ) {
    for ( int j = 0; j < vPhasesClusters[z].size() && bFound == false; j++ ) {
      for ( int k = 0; k < vPhasesClusters[z][j].size() && bFound == false; k++ ) {
        if ( vPhasesClusters[z][j][k] == vPhaseFlynns[i][0] ) {
          vCurrentArea.push_back( vPhasesClusterAreas[z][j] );
          bFound = true;
```

However if they belong to the same phase their area change is added together. There is also a check to make sure there were no errors in the previous simulation. If, like for the triple node in the example,

there are two Flynns of the same phase neighbouring the node it is checked whether the stored original area for both Flynns is the same. If that is not the case something went wrong in the previous part of the process and an error message is displayed as well as some functions are called to correct that problem. I will not explain these check functions at this stage since they will be explained later on. In the end the function determined the area change of the Flynns. It has read the original area of the cluster the Flynns belong to from the Flynn attribute and it has read the Flynns current area from the clusterTracking class variable. After some more checks, which make sure that all vectors contain the required information, all of these values are set in relation to each other and scaled by values from the config file by equation 2.

$$AE = \alpha \cdot \left(\frac{(A_t + A_{\Delta t}) - A_0}{A_0}\right)^{\beta} \qquad (2)$$

```
vClusterAreaEnergy.push_back ( dMultiplierA * pow ( fabs( ( ( vCurrentArea.at( i ) + ( vPhaseClusterAreaChange.at( i ) ) ) -
vClusterArea.at( i ) ) / vClusterArea.at( i ) ), dMultiplierD ) );
```

The area energy (AE) equals the current area (A_t) including the small change by moving to one of the trial positions ($A_{\Delta t}$) compared to its original area (A_0) normalized by the original area. That value is scaled by a scalar (α) and also by an exponent (β) both can be set in the config file and are used to set the sensitivity of the experiment towards phase area changes and also to adjust the energy level to the general energy level in the simulation to prevent larger changes in the beginning of the simulation.

Now if energy tracking is set for one node in the config file. These values will be written to "1.txt" once that node is under investigation.

A vector containing the area energy values for each phase is returned to the function "returnClusterAreaEnergy" which will just add all values in the vector up and return the resulting scalar to "GGMoveNode".

```
for ( int i = 0; i < vClusterAreaEnergy.size(); i++ )
 dClusterAreaEnergy += vClusterAreaEnergy.at ( i );

return dClusterAreaEnergy;
```

After both energies have been calculated for all four trial positions. The resulting total energy, which is just the sum of both energies, at all trial positions will be passed to the next function called "GetMoveDir".

"GetMoveDir" and the function it calles ("MoveDNode" or "MoveTNode") are unchanged in regard to their original versions in the GBM code by J. Becker [(1)] except that the mobility for the phase boundaries is read from the config file. Basically they are calculating the energy field surrounding the node under investigation. From that a movement direction is derived which is then modified by the mobility for the individual segments in contact with the node. In the end a movement vector for the node is returned which in regard to its current position will result in the new position for the node.

```
        ElleSetNodeChange(0);
        ElleCrossingsCheck( ran.at( j ), & newxy );
        if (ElleNodeChange() != 0)
          clusters.updateClusters();
```

The node will be moved by "ElleCrossingsCheck" which will move the node to the new position and check for any topology problems which can result from that movement. To make this code faster a change has been implemented to the base code. Before each function which could alter the topology setting a call of "ElleSetNodeChange(0)" will set a variable to 0. Now if in the following function the topology is actually altered that variable will be changed. Otherwise it will stay 0. If it is still 0 afterwards nothing has to be done. If the topology was altered the whole cluster class has to be updated since clusters could have been changed. This is done by a call of "updateClusters".

```
void clusterTracking::updateClusters( void )
{
  clusterTracking::findClusters();
  clusterTracking::findSplit();
  clusterTracking::findMerge();
  clusterTracking::checkDoubleClusterAreaLoop();
}
```

"updateClusters" consists of four sub function. First there will be a call of "findClusters" which I already described during the discussion of the constructor. So all clusters will be determined and stored in a triple vector. Since the current setup might be different from the previous one before the topology change the code checks for split events afterwards. That means part of the cluster might have been cut off due to boundary movement and the previous single cluster is now divided in two separate parts which are not connected anymore. That is done with the help of the cluster areas which are stored in one of the set Flynn attributes for each Flynn. Each Flynn in one cluster has got exactly the same number. If two clusters would have had exactly the same number during construction of the cluster in the first step, their areas would have been shifted by a small amount mentioned before to prevent that sort of problem. Now at the current stage of the simulation the only way for two clusters to have exactly the same area number is that there was a split.

```
for ( int z = 0; z < vPhasesClusters.size(); z++ ) {
  for ( int i = 0; i < vPhasesClusters[z].size(); i++ ) {
    ElleGetFlynnRealAttribute( vPhasesClusters[z][i][0], &dArea, iFlynnCluster );

    vSplitClusters.clear();

    for ( int j = 0; j < vPhasesClusters[z].size(); j++ ) {
      vSplitClusterFlynns.clear();
      for ( int k = 0; k < vPhasesClusters[z][j].size(); k++ ) {
        ElleGetFlynnRealAttribute( vPhasesClusters[z][j][k], &dAreaCheck, iFlynnCluster );
        if ( dArea == dAreaCheck ) {
          vSplitClusterFlynns.push_back( vPhasesClusters[z][j][k] );
        }
      }
      if ( vSplitClusterFlynns.size() > 0 ) {
        vSplitClusters.push_back( vSplitClusterFlynns );
      }
    }
    // Wenn mehr als ein Cluster mit Flynns mit der gleichen Fläche gefunden wurde
    // --> Der Cluster hat sich geteilt --> Flächen neu verteilen.
    // (Ein Cluster bedeutet der Cluster selbst wurde gefunden)
    if ( vSplitClusters.size() > 1 ) {
      clusterTracking::resolveSplit( vSplitClusters );
```

This is exactly what the “findSplit” function does. It checks whether there are two entries in the Cluster vector which Flynns have the same are number. If it detects such an event it will call the function “resolveSplit”.

```
ElleGetFlynnRealAttribute( vSplitClusters[0][0], &dSplitClusterNewArea, iFlynnCluster );
cout << "Cluster with same areanumber detected.... (SPLIT) " << dSplitClusterNewArea << " |";
dSplitClusterAreaComplete = 0;
for ( int j = 0; j < vSplitClusters.size(); j++ ) {
  dSplitClusterAreas[j] = 0;
  cout << "| ";
  for ( int k = 0; k < vSplitClusters[j].size(); k++ ) {
    dSplitClusterAreas[ j ] += ElleRegionArea( vSplitClusters[j][k] );
    cout << vSplitClusters[j][k] << " ";
  }
  dSplitClusterAreaComplete += dSplitClusterAreas[ j ];
}
cout << "|" << endl;
for ( int j = 0; j < vSplitClusters.size(); j++ ) {
  // Calculate Ratio for that part of the split Cluster (Split Part / Current Complete Area) --> For the Ratio calculation the
old Area is not used.
  dSplitClusterRatio = dSplitClusterAreas[ j ] / dSplitClusterAreaComplete;
  // Calculate New Area with the OLD Area and the calculated Ratio
  ElleGetFlynnRealAttribute( vSplitClusters[j][0], &dSplitClusterNewArea, iFlynnCluster );
  dSplitClusterNewArea *= dSplitClusterRatio;
  // Write new Area in that part of the Flynn.
  for ( int k = 0; k < vSplitClusters[j].size(); k++ ) {
    ElleSetFlynnRealAttribute( vSplitClusters[j][k], dSplitClusterNewArea, iFlynnCluster );
```

This function does nothing else than to adjust the area number for both parts of the old cluster. It will calculate the ratio of both parts with the help of their current areas in relation to their total current area. The original area is then divided with these ratios and the new values are assigned to each new cluster individually.

```
for ( int z = 0; z < vPhasesClusters.size(); z++ ) {
  for ( int i = 0; i < vPhasesClusters[z].size(); i++ ) {
    lNotMatchingAreas.clear();
    ElleGetFlynnRealAttribute( vPhasesClusters[z][i][0], &dArea, iFlynnCluster );
    for ( int j = 1; j < vPhasesClusters[z][i].size(); j++ ) {
      ElleGetFlynnRealAttribute( vPhasesClusters[z][i][j], &dAreaCheck, iFlynnCluster );
      // if the Cluster Areas of the Flynn ain't match the first one... --> Merge?
      if ( dArea != dAreaCheck ) {
```

```
      lNotMatchingAreas.push_back(dAreaCheck);
    }
  }
  // all double Entries have to be deleted... --> problem if two of the merged clusters had the same areas...
  lNotMatchingAreas.sort();
  lNotMatchingAreas.unique();

  if ( lNotMatchingAreas.size() > 0 ) {
    lNotMatchingAreas.push_back(dArea);
    clusterTracking::resolveMerge( z, i, lNotMatchingAreas );
```

The next sub function called by "updateClusters" is "findMerge". Find merge will detect two clusters that have merged together during grain boundary movement. The two clusters which used to have two separate entries in the Clusters vector will now end up in only one entry. However their area numbers are still different because they used to be two separate clusters. This is exactly how the "findMerge" function works. It will loop through every entry in the Clusters vector and check whether all Flynns have got the same area number. If they don't there was a merge during the last topology check.

```
double dMergedArea = 0.0;
// calculate new cluster area (just add the old areas together)

cout << "Cluster has different Clusterflynns... (MERGE) ";
while ( lNotMatchingAreas.size() > 0) {
  cout << lNotMatchingAreas.back() << " ";
  dMergedArea += lNotMatchingAreas.back();
  lNotMatchingAreas.pop_back();
}
cout << ":: " << dMergedArea << endl;
// set new area for ALL flynns in that cluster!
for ( int j = 0; j < vPhasesClusters[z][i].size(); j++ ) {
  ElleSetFlynnRealAttribute( vPhasesClusters[z][i][j], dMergedArea, iFlynnCluster );
```

"resolveMerge" will be called which adds all original areas stored in the Flynn attribute from all Flynns in the cluster. Afterwards all duplicate entries are deleted. That way the code has got one entry for each cluster that has merged into the new combined cluster. In case two clusters merged together there are two entries. With three there is one more and so on. All of these different original areas are then summed up and stored in the set Flynn attribute for each Flynn of the new combined cluster updating the old values.

The last function called by "updateClusters" is another call of "checkDoubleClusterAreaLoop". I already discussed this function during the explanation of the constructor. The only thing it does is to loop through all entries in the Clusters vector and check whether there are two entries which have exactly the same area number. That could have happened during the adjustment of the split or merge functions. If it happens their values are adjusted by a small amount. To prevent the overall area from changing one cluster gets assigned a slightly larger number while the other a slightly smaller number.

After all four functions are completed the main loop continues. Next are calls for “ElleCheckDoubleJ” and “ElleCheckTripleJ”. While the DoubleJ function is trivial and only inserts or deletes a double node if the distance to the neighbouring nodes is too large or to small respectively, the TripleJ function is more complicated. What has to be said is that in a correct Elle topology there can only be Flynns with at least two triple nodes otherwise the Flynn ends up to be a Flynn inside another Flynn which leads to problems. Now “ElleCheckTripleJ” induces triple switches between two triple nodes which are too close together. Under certain conditions that could lead to a single triple node Flynn in another Flynn. To prevent that, the other Flynn is also split and essentially a new Flynn is created (Fig 4).

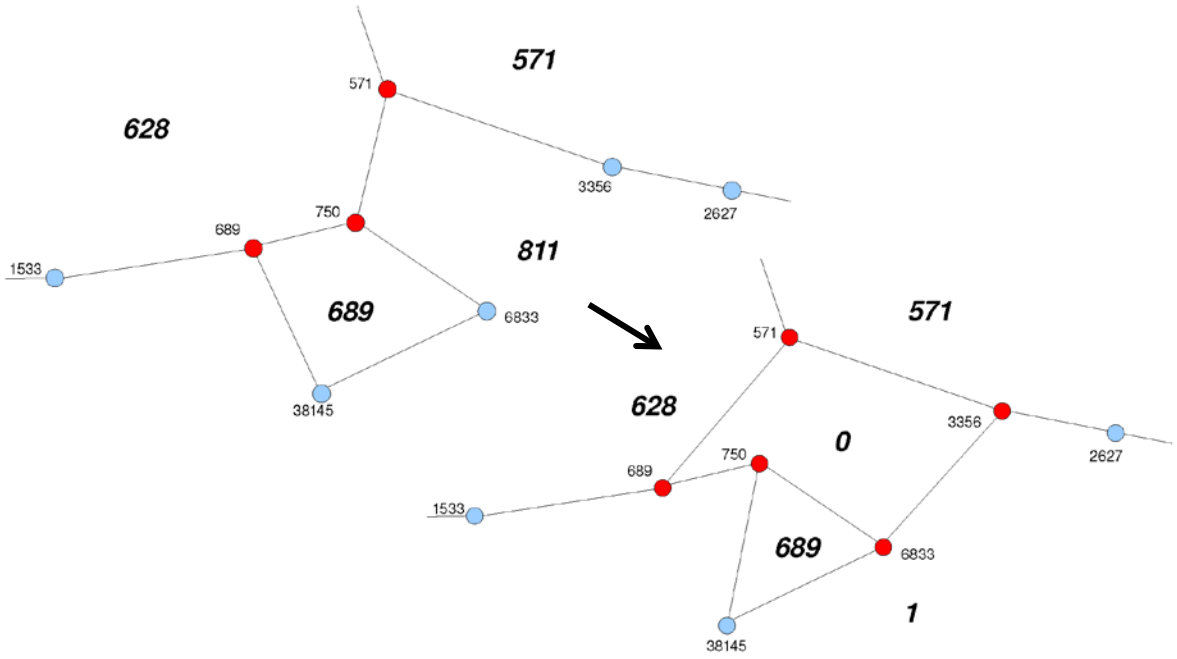


FIG 4 FLYNN 689 IS SMALL AND SHRINKING. AT SOME STAGE “ELLECHECKTRIPLEJ” WILL SWITCH NODE 689 AND 750. SINCE THAT WOULD RESULT IN A FLYNN IN FLYNN (689 IN 811) A SPLIT IS INDUCED WHICH SPLITS FLYNN 811 INTO 0 AND 1. FLYNN 689 IS NOW A FLYNN WITH ONLY THREE NODES AND WILL BE DISREGARDED BY “ELLECHECKTRIPLEJ” SINCE IT IS VERY LIKELY THAT IT WILL DISAPPEAR SOON. (FIGURE AFTER EVANS, L. (PC))

This procedure is messing with Flynn statistics and in some cases the number of Flynns could be increasing even if there is no process in the experiment which could induce that.

```
if ( iMinTjs > 2 ) {
  int iNeighbours[3], iFlynns[3];
  int iNodeCount = 0;
  int * iNodes = NULL;
  int iTripleCheck;

  ElleNeighbourNodes( j, iNeighbours );
  for ( int k = 0; k < node_type; k++ ) {
    ElleNeighbourRegion( j, iNeighbours[k], &iFlynns[k] );
    ElleFlynnNodes( iFlynns[k], &iNodes, &iNodeCount );
    iTripleCheck = 0;
    for ( int l = 0; l < iNodeCount; l++ ) {
      if ( ElleNodeIsTriple ( iNodes[l] ) ) {
        iTripleCheck++;
      }
    }
```

```
  if ( iTripleCheck < iMinTjs ) {
   fstream fTripleFile;
   fTripleFile.open ( "FailedTripleSwitches.txt", fstream::out | fstream::app);
   if (fTripleFile.is_open()) {
    fTripleFile << i << ": " << j << "{" << iTripleCheck << " (" << iNeighbours[k] << ") […]
    fTripleFile.close();
   }
   break;
  }
  free( iNodes );
  iNodes = NULL;
 }
 if ( iTripleCheck >= iMinTjs ) {
  ElleSetNodeChange(0);
  ElleCheckTripleJ( j );
  if (ElleNodeChange() != 0)
   clusters.updateClusters();
```

To prevent that there is a switch in the loop which checks how many triple nodes a Flynn has and it only calls “ElleCheckTripleJ” if the Flynn has more triple nodes than set in the beginning of the file (config section). In general this switch is deactivated and all triple nodes will be checked with “ElleCheckTripleJ”. Basically this is the end of the main loop and it will continue with the next time step.

There are additional statistics functions to track melt evolution with unodes if that has been set in the config file. For phase tracking values in the config file that are equal or larger to 0 phase tracking will be done for the phase equaling that number. That means there are two different values stored in the unode attributes. U_ATTRIB_A will hold information about how many time steps of the total amount of time steps that unode has been part of the tracked phase. U_ATTRIB_B will hold information on how often that node has been converted from any other phase to the tracked phase. U_ATTRIB_C is just a variable which is used to save the phase the unode currently belongs to. This way the code is able to compare that value with the value of the next step which is necessary to determine the amount of phase changes.

If the phase tracking value equals “-1” the unodes are used in a different way. All unodes can contain information which could be concentration in one unode attribute which is set in the beginning of the file. This concentration is not diffused. Every step it is determined which “concentration” belongs to which phase. The values are summed up for each phase separately and written to the file. It was used to check how the diffusion code for unodes of Gomez-Rivaz, E. works together with gbm_pp.

If the phase tracking value is less than -1 the phase tracking is deactivated.

1.4.4 FFT IMPLEMENTATION

During a visit in Barcelona Albert Griera and me tried to combine my code for two or more phase grain boundary migration with his grain boundary migration modification to take dislocation density from the fft viscoplastic deformation into account. Summarized it wasn't a big problem. We just had to make sure that the storage variables are not used twice and we had to copy some functions from one gbm version to the other. In general both versions calculate a different additional energy. For my process this would be the area energy to keep the phase areas constant and for his process this would be the internal energy in grains as a result from the dislocation density. Both functions used the same story slots because they were very limited by this time. Also using both processes like it is the normal Elle procedure wasn't an option because both actually represent the same process with different additions. Since our attempt to adjust both codes they already have been further developed and that is why further thought has to be spent to make use of both codes again. However since the newest version of the poly phase grain boundary migration code mainly uses the Flynns as storage containers and also a recent modification to the basecode allows more storage variables to be used, this shouldn't be a big deal. All storage variables for gbm_pp can be set in the beginning of the source code file. Copying the additional internal energy functions along with a few other adjustments should be the main part.

1.4.4.1 MULTITHREADING

Since the calculation of the internal energy during the gbm process is very time consuming we also spent thoughts about making this process multithreading. With other processes this can be done easily by splitting the available data in 2 or more parts and just starting another thread with the script. Later on the results have to be combined again. For gbm this is not that easy because the data points are actually dependent on each other. Moving one immediately affects the next calculation. Splitting the points would lead to different results. What is possible however is starting a thread for every trial position calculation because during the calculation of the trail position nothing in the data structure is changed. This way not the complete process is multithreading, the most intensive part is however. We made use of the boost library to start one separate thread for each of the four trial positions. At first it seemed to work alright and we measured a speed increase of up to 300%. During the test simulations we experienced random crashes and our investigations didn't get to the bottom of that. Our conclusion was that they were related to the fact that we didn't copy the Elle data arrays for each thread and they were all using the same array to get the data for their calculations. Since we didn't want to use four times as much memory for this process further development was stopped.

1.4.4.2 GENERAL SETUP

A few more files are required to run a simulation which combines gbm_pp and the fft viscoplastic deformation. The main loop is commonly stored in a file called launch.sh or similar. Usually the loop starts with one step of fft deformation. For that the elle file in the directory has to be prepared. Albert wrote a very helpful description on how to use and prepare files for the use with fft called “Introduction to FFT-ELLE”. It can be found in the fft directory on the CD but should also be part of each Elle download from Sourceforge with cvs. Important parameters in the elle file are dimensions and temperature as well as the phase number for the individual Flynns.

The FFT process needs the files make.out, ice3d.sx, ppc.in, ppc.dim and temp.out. Make.out contains the information of the individual points. Ice3d.sx is the crystal file and contains information about the slip systems available in ice. For different materials a different file is necessary. Ppc.in contains the settings for the simulations like strain rate. Temp.out is a dummy file and ppc.dim only needs modifications of the resolution should be changed (default is 256x256x1).

Some details about ice3d.sx are the tau0xf and tau0xb values. They determine the energy needed for activation of theses slip systems. For the pyramidal and prismatic slip planes they are set to 20 compared to 1 for the basal plane. This means it takes 20 times more energy to activate them compared to the basal plane.

A few settings can be adjusted in the ppc.in file. The number of nodes has to be adjusted to the resolution. 256x256x1 = 65536. Then the name of the output file and the name of the crystal file can be specified. Next are the boundary conditions. Because we assume plain strain we only need $e_{xx} = \frac{\gamma}{2}$ and $e_{yy} = -\frac{\gamma}{2}$ for pure shear or $e_{xy} = \gamma$ for simple shear. After that we set the step size (in seconds) and with imax the maximum number of iterations can be set. More takes longer but is more accurate. Default is about 400. The last two numbers are the length scale which should be the same like in the elle files and a value to estimate dislocation density, the Burgers vector for basal slip.

Additionally I will provide some information about reading output data if required from the make.out and tex.out files. The make.out file is split in three parts. First part is the first line and contains the number of grains. Second part contains the Euler angles of the grains followed by the FFT grain number. The last part contains the Euler angles of the unodes followed by the FFT point location, the Flynn number that contains that point and the Phase ID. The tex.out file can basically be read like columns. The lines represent the unode data points. The first three columns are the Euler angles again. The next one is the Phase ID followed by the strain and stress values in the next two columns. The next number represents the basal slip activity which is followed by that for the prismatic slip. Next are

geometrically necessary dislocation density and the statistical dislocations. The last two columns are the unode points and the grain number in FFT again.

```
./fft256/fft_256                                          ~ Run FFT
 cp temp-FFT.out temp.out                                 ~ Copy an output file
 fft2elle -i elle2fft001.elle –n                          ~ Conversion from FFT results to the elle format.
 reposition -i fft2elle.elle –n                           ~ Check all nodes and reposition them if necessary.
 importFFTdata_user_DDsum -i repos.elle -u 4 5 -n         ~ Import FFT results to variables in the ELLE file
# FFT finished

shiftphase -i fft_out.elle -s 1 -f 1 -n -u -1             ~ In case the processes use different phase numbers
elle_gg -i shiftphase001.elle -s 1 -f 1 –n                ~ Helps preventing problems sometimes.

# 1. Step of NUCLEATION

cp growth001.elle tmp.elle                                ~ Copy the output file
dislocden_rx -i tmp.elle -n -u 1e13 1 50                  ~ Run Nucleation based on dislocation density
tricky_vs02 -i dislocden_rx.elle -s 1 -f 1 –n             ~ Repair some unnodes in case of boundary changes

# 1. Step of GBM

full_gbmunodes -i tricky.elle -s $gbmsteps -f 1 -n -u 0   ~ Run gbm
cp gbmunodes0$gbmsteps.elle gbm$gbmsteps_ok1.elle         ~ Copy the output file
mv gbmunodes0$gbmsteps.elle splitSGG.elle                 ~ Rename the output file

# 1. STEP of SGG
splitflynnSGG -i splitSGG.elle -n -u 4                    ~ Run recovery as 4 threaded process
parallelSGG -i splitSGG.elle -s $sggsteps -n -u 15 1 &
parallelSGG -i splitSGG.elle -s $sggsteps -n -u 15 2 &
parallelSGG -i splitSGG.elle -s $sggsteps -n -u 15 3 &
parallelSGG -i splitSGG.elle -s $sggsteps -n -u 15 4 &

while [ ! -e 1.unodes ]; do
sleep .1
done

while [ ! -e 2.unodes ]; do
sleep .1
done

while [ ! -e 3.unodes ]; do
sleep .1
done

while [ ! -e 4.unodes ]; do
sleep .1
done                                                      ~ Wait until all threads are finished

importparallelSGG -i splitSGG.elle -n -u 4 $sggsteps      ~ Put the files together again

while [ ! -e pSGG.0$sggsteps.elle ] ; do
sleep .1
done

checkangle -i pSGG.0$sggsteps.elle -s 1 -f 1 -n -u 0.4 1  ~ Check for some small angles
elle_gg -i checkangle001.elle -s 1 -f 1 –n                ~ Helps preventing problems sometimes
mv pSGG.0$sggsteps.elle sgg$sggsteps_ok1.elle             ~ Rename output file
rm splitSGG.elle *.unodes checkangle001.elle              ~ Clean temporary files
shiftphase -i growth001.elle -s 1 -f 1 -n -u 1            ~ Shift phases back for FFT.
```

Example of a launch.sh script file with a few explanations on what is called with each line.

1.5 PERSONAL MINI PROGRAMS

During the last 4 years I wrote a few small programs which help with the daily workflow of the simulations and results. Most of them are related to easier data handling and statistic processing of the experiments.

1.5.1 JR-STATS

JR-stats is a modification of the already existing stats routine. It needs the userdata options. The first paramteter of the userdata array tells the routine which of the stats functions it should call. At the moment there are six different ones.

1) Is very simple. It doesn't need additional parameters. It just counts all Flynns and calculates the average area (1 / number of Flynns)
2) Also doesn't need additional parameters. It writes 2 files. In area_phase_side.txt it writes out every grain along with its area, phase number and amount of neighbouring grains. In grains_splitgrains.txt it just writes one line which says how many grains there are and how many of them have been split.
3) This is the old stat function. It does exactly the same thing.
4) Needs one additional parameter as the second userdata parameter which tells the function how many phases are in the experiment. For two phases the call would be –u 4 2. It counts the number of grains and their area for each phase separately and writes them out in log.txt.
5) This function is a bit more complicated and was only useful for a specific analysis of bubble migration speed in an idealised microstructure. Basically it calculates the distance of the centre point of a grain towards a position along with some angles.
6) Creates a script file to call stats on many different elle files. This has been replaced by the python script.

1.5.2 JR_COLLECTION

This little program helps with random allocation of attributes. When run, it will ask a few questions in the command line to carry out specific functions. It can allocate the expand attribute randomly as well as set random c-axis orientations. The phase and viscosity settings can be set (different options) or the Flynn numbers can be saved to an attribute of choice. Also the phases of specific Flynns can be adjusted (min T nodes, max T nodes, max area). The advantage of doing it with this program is that the results can be seen immediately if the colouring of the Flynns is set previous to running the program. For that

the program has to be loaded with an elle file. All visualisation settings have to be set and then the simulation has to be started. On the first step it will ask all the necessary questions in the command line window.

1.5.3 ELLE FILE CREATOR

This little tool creates elle files with a perfect 120° microstructure. When run it will ask some questions about how many hexagons and the file name. After that it writes out an elle file with the default header settings. It also only creates the triple nodes. The double nodes have to be inserted by running one step of elle_gg or similar programs.

1.5.4 PYTHON SCRIPTS

Three small python scripts to make life easier.

Rename.py calls the unix rename function on all gbm_pp file names in a directory. This can be changed in the script. Doing that prevents sorting problems with other programs since the default elle numbering starts with three numbers. More than 999 steps result in four numbers and so on. Also having letters like the process name in the file name sometimes prevent proper sorting. After the rename is complete all files have a six digits file name by default. If an argument is supplied they will have a seven digits file name. It is advisable to use this script before using stats.py or other scripts.

Stats.py basically takes the same parameters as the jr-stats function. It will call the latter on all elle files in the directory sequentially, renaming the log files if necessary to prevent them from being overwritten.

Extract.py is helpful after fft experiments. Usually these experiments are compressed after each step to save disc space and to keep order with all the different output files. The compressed files are saved in different directories for every step. If all the elle files from the archives are needed for analysis it can be tedious to extract them manually. This is what this script does. Extract all elle files from the archives and put them together into a separate folder, renaming them according to their step number.

2 REFERENCES

1 Becker, J. K., Bons, P. D. & Jessell, M. W. A new front-tracking method to model anisotropic grain and phase boundary motion in rocks. *Computers & Geosciences* **34**, 201-212, (2008).

2 Roessiger, J., Bons, P. D. & Faria, S. H. Influence of bubbles on grain growth in ice. *Journal of Structural Geology* **in press**, (2012).

APPENDIX 5

PROCESS CODE

CONTENTS

1 SPLIT 2

1.1 HEADER: SPLIT2.H

```
#ifndef SPLIT2_ELLE_H_
#define SPLIT2_ELLE_H_
#include <vector>
#include <algorithm>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include "flynnarray.h"
#include "attrib.h"
#include "nodes.h"
#include "file.h"
#include "display.h"
#include "check.h"
#include "error.h"
#include "runopts.h"
#include "init.h"
#include "general.h"
#include "stats.h"
#include "update.h"
#include "interface.h"
#include "polygon.h"

typedef struct
{
  int x, y;
  double error;
} DEVIATION;

int Init_Split2(void);
int intsplit2(void);
int directsplit2(int flynn, int start, int end, int *c1, int *c2);
int randomsplit2(int flynn, double mcs, int *c1, int *c2);
int directionsplit2(int flynn, double x, double y, double mcs, int *c1, int *c2);
int nodes2childs(int **id, int num_nodes, int start, int end, int **child1, int **child2, int *nchild1, int *nchild2);
double areacheck(int **nodes, int num_nodes);
int intersectioncheck(int **child, int nchild);
int flynnsplit2(int flynnindex, int start, int end, int **child1, int **child2, int *nchild1, int *nchild2, int **c1, int
**c2);
int assignstruct(int **id, double dir, int num_nodes, int *possis);
void sortstruct(DEVIATION items[], int left, int right);

#endif /* SPLIT2_ELLE_H_ */
```

1.2 CODE: SPLIT2.CC

```
#include "split2.elle.h"

using std::vector;

// this is IMPORTANT. A flynn has to have a minimum of defined double nodes
// otherwise it won't split. Dependent on switch distance it influences the
// min flynn size.
#define MINDNODES 2
// this defines whether the 2nd try approach (step 10 for direction and randomsplit) is used or not
// set to 0 if you don't want to use it.
#define SECONDTRY 1

UserData userdata;

FILE *split;

DEVIATION *dev;

int Init_Split2(void)
{
    int err=0;
    int max, maxf, n;
    char *infile;

    ElleReinit();
    ElleSetRunFunction(intsplit2);

    infile = ElleFile();
    if (strlen(infile)>0) {
        if (err=ElleReadData(infile)) OnError(infile,err);

        ElleAddDoubles();
    }
}

int intsplit2(void)
{
  int splittype, flynn, start, end, c1, c2, check=0;
```

```
39      double dx, dy, mcs; //mcs=min_child_size
40
41        ElleUserData(userdata);
42        splittype = (int)userdata[2];
43        flynn = (int)userdata[0];
44        //get the min_child_size as fraction of the parent grain.
45        //If userdata returns 1 than the global or MINERAL specific data is retrieved.
46        mcs = (double)userdata[1];
47        if (mcs==1)
48         mcs=ElleFindFlynnMinArea(flynn);
49
50        //if splittype is out of range, define random split as default
51        if (splittype<1 || splittype>3)
52         splittype=1;
53
54        //for random split define random dx and dy and split
55        if (splittype==1) {
56       check = randomsplit2(flynn, mcs, &c1, &c2);
57      }
58        //for direction split get the dx and dy directions from userdata and split
59        if (splittype==2) {
60         dx = (double)userdata[3];
61         dy = (double)userdata[4];
62         check = directionsplit2(flynn, dx, dy, mcs, &c1, &c2);
63        }
64        //for direct split get the start and end nodes from userdata and split
65        if (splittype==3) {
66       start=(int)userdata[3];
67       end=(int)userdata[4];
68       directsplit2(flynn, start, end, &c1, &c2);
69       check=1;
70      }
71
72        //printf("Childs: %d & %d\n", c1, c2);
73        if (check==1 || check==3) {
74       EllePromoteFlynn(c1);
75       EllePromoteFlynn(c2);
76       ElleRemoveFlynn(flynn);
77       ElleAddDoubles();
78        }
79        ElleUpdate();
80
81    }
82
83    int directsplit2(int flynn, int start, int end, int *c1, int *c2)
84    {
85      int *nodes=0, num_nodes;
86      int *child1=0, *child2=0, nchild1=0, nchild2=0;
87
88      ElleFlynnNodes(flynn, &nodes, &num_nodes);
89      nodes2childs(&nodes, num_nodes, start, end, &child1, &child2, &nchild1, &nchild2);
90      flynnsplit2(flynn, start, end, &child1, &child2, &nchild1, &nchild2, &c1, &c2);
91
92      //EllePromoteFlynn(c1);
93      //EllePromoteFlynn(c2);
94      //ElleRemoveFlynn(flynn);
95
96      free(nodes);
97      free(child1);
98      free(child2);
99    }
100
101   int randomsplit2(int flynn, double mcs, int *cc1, int *cc2)
102   {
103     int i, c1, c2;
104     double x, y;
105
106     x=ElleRandomD();
107     y=ElleRandomD();
108     for (;x==0.0 && y==0.0;) {
109       x = ElleRandomD();
110       y = ElleRandomD();
111     }
112     x*=2;
113     x-=1;
114     y*=2;
115     y-=1;
116     i = directionsplit2(flynn, x, y, mcs, &c1, &c2);
117     *cc1=c1;
118     *cc2=c2;
119     return i;
120   }
121
122   //#define MINAREA 0.0002 // flynn has to be larger than that to actually be able to split.
123
124   int directionsplit2(int flynn, double x, double y, double mcs, int *c1, int *c2)
125   {
126     int i, j=0, check=0, *id=0, maxnint, num_nodes, start=0, end=0, starti=0, endi=0, *child1=0, *child2=0, nchild1=0,
127   nchild2=0, possis, nd=0;
128     double dir, min_area, test_area, maxn;
129     vector<int> seq;
130
131     // find all the Nodes of a specified flynn
132     ElleFlynnNodes(flynn, &id, &num_nodes);
133
134     nd = SECONDTRY;
135
136     // don't know if this is really good, but it helps to randomize the split origins....
137     maxn = ElleRandomD();
138     maxn *= num_nodes;
139     maxnint = (int)maxn;
```

```
140       ElleSetFlynnFirstNode(flynn, *(id+maxnint));
141       free(id);
142       ElleFlynnNodes(flynn, &id, &num_nodes);
143
144
145       // set minimum area
146       min_area = areacheck(&id, num_nodes);
147       min_area *= mcs;
148
149       //calculate direction relative to x-axis
150       dir=atan(y/x);
151
152          // assignt the struct of arrays explained in step 4
153          if ((check=assignstruct(&id, dir, num_nodes, &possis))==1) {
154            // step 5 use quicksort to sort the struct
155            sortstruct(dev, 0, possis-1);
156            //if ((test_area=areacheck(&id, num_nodes))>MINAREA) {
157              // start from the first to the last entry in the deviation struct
158              for (j=0,i=0;j<possis && i==0;j++) {
159                start=dev[j].x;
160                end=dev[j].y;
161                if ((check=nodes2childs(&id, num_nodes, start, end, &child1, &child2, &nchild1, &nchild2))==1)
162                  //check min area of child 1
163                  if ((test_area=areacheck(&child1, nchild1))>=min_area)
164                    //if ok, check min area of child 2
165                    if ((test_area=areacheck(&child2, nchild2))>=min_area)
166                      //if ok, check intersections of child 1 with split direction
167                      if (intersectioncheck(&child1, nchild1))
168                        //if ok, check intersections of child 2
169                        if (intersectioncheck(&child2, nchild2)) {
170                            flynnsplit2(flynn, start, end, &child1, &child2, &nchild1, &nchild2, &c1, &c2);
171                            //printf("Successfully split flynn %d\n", flynn);
172                            i=1;
173                        }
174              }
175              if (j==possis && i==0 && nd == 1) {
176                min_area /= 2;
177                //nd=1; // marker for the 2nd try
178                //split=fopen("split.txt", "a");
179                //fprintf(split,"nd-try startet\n");
180                //fclose(split);
181                for (j=0,i=0;j<possis && i==0;j++) {
182                  start=dev[j].x;
183                  end=dev[j].y;
184                  if ((check=nodes2childs(&id, num_nodes, start, end, &child1, &child2, &nchild1, &nchild2))==1)
185                    //check min area of child 1
186                    if ((test_area=areacheck(&child1, nchild1))>=min_area)
187                      //if ok, check min area of child 2
188                      if ((test_area=areacheck(&child2, nchild2))>=min_area)
189                        //if ok, check intersections of child 1 with split direction
190                        if (intersectioncheck(&child1, nchild1))
191                            //if ok, check intersections of child 2
192                            if (intersectioncheck(&child2, nchild2)) {
193                                      flynnsplit2(flynn, start, end, &child1, &child2, &nchild1, &nchild2, &c1, &c2);
194                                      //printf("Successfully split flynn %d\n", flynn);
195                                      i=3;
196                            }
197                }
198              }
199       //      for(i=0; i<possis; i++)
200       //        printf("DEV%d: %d - %d : %f\n", i, dev[i].x, dev[i].y, dev[i].error);
201       //      printf("Possies: %d\n", possis);
202              free(dev);
203              free(child1);
204              free(child2);
205            //} else {
206              //printf("ERROR: flynn too small to split\n");
207            //}
208          } else {
209            if (check==2) // this is to not count split attempts of too small grains as errors
210              i=2;
211            else
212              printf("ERROR: split2 completely failed: assignstruct error\n");
213          }
214
215       free(id);
216
217       if (i==1)  // successful split
218         return 1;
219       else if (i==2)  // too small grain
220         return 2;
221       else if (i==3)// successful split after 2nd try with half min_area
222         return 3;
223       else
224         return 0;// error
225     }
226
227     int nodes2childs(int **id, int num_nodes, int start, int end, int **child1, int **child2, int *nchild1, int *nchild2)
228     {
229       int i, j, starti, endi, temp, *iptr;
230       /* if a matching direction is found
231        * the nodes are are written into two possible child arrays which can be
232        * further investigated.
233        * start node of the possible split is always element 0
234        * and end node always the last element
235        * all the other nodes are arranged between those two
236        *
237        * returns 1 if successful and 0 if not successful.
238        */
239
240       // find the position of the start and end nodes within the array
```

```
241       for (i=0; i<num_nodes;i++) {
242         if (start==*(*id+i))
243           starti=i;
244       }
245       for (i=0; i<num_nodes;i++) {
246         if (end==*(*id+i))
247           endi=i;
248       }
249       //printf("%d, %d\n", start, end);
250
251       // Exchanges starti and endi if starti is bigger than endi
252       // this is needed because the first child doesn't check for start/end overstep
253       if (starti>endi) {
254         temp=starti;
255         starti=endi;
256         endi=temp;
257       }
258
259       // child1
260       // count elements for child 1 and sets *nchild1 accordingly
261       for (i=starti,j=0;i<=endi;i++,j++) {
262         ;
263       }
264       *nchild1=j;
265
266       // writes all the elements of child1 into the child1 array
267       if ((*child1 = (int *)malloc(*nchild1 * sizeof(int)))==0) {
268         printf("ERROR: nodes2childs: Malloc_Err: child1\n");
269         return 0;
270       }
271       for (i=starti,j=0,iptr=*child1;i<=endi;i++,j++)
272         iptr[j] = *(*id+i);
273
274
275       // child2
276       // do the same for child2
277       // except it starts at endi to the end and continues at 0 to stari
278       for (i=endi,j=0;i<num_nodes;i++,j++) {
279         ;
280       }
281       for (i=0; i<=starti;i++,j++) {
282         ;
283       }
284       *nchild2=j;
285
286       if ((*child2 = (int *)malloc(*nchild2 * sizeof(int)))==0) {
287         printf("ERROR: nodes2childs: Malloc_Err: child2\n");
288         return 0;
289       }
290       for (i=endi,j=0,iptr=*child2;i<num_nodes;i++,j++)
291         iptr[j] = *(*id+i);
292       for (i=0;i<=starti;i++,j++)
293         iptr[j] = *(*id+i);
294
295       return 1;
296     }
297
298     double areacheck(int **nodes, int num_nodes)
299     {
300       /* This one is copied from elsewhere except
301        * I have commented out the ElleFlynnNodes function and pass the these
302        * values as function arguments instead.
303        * It returns the area between the nodes passed.
304        */
305         int j; //*id=0;
306         double area, *coordsx=0, *coordsy=0, *ptrx, *ptry;
307         Coords xy,prev;
308
309         //ElleFlynnNodes(poly,&id,&num_nodes);
310         if ((coordsx = (double *)malloc(num_nodes*sizeof(double)))== 0)
311          printf("ERROR: areacheck: Malloc_Err: coordsx\n"); //OnError("ElleRegionArea",MALLOC_ERR);
312         if ((coordsy = (double *)malloc(num_nodes*sizeof(double)))== 0)
313          printf("ERROR: areacheck: Malloc_Err: coordsy\n");  //OnError("ElleRegionArea",MALLOC_ERR);
314         ElleNodePosition(*(*nodes),&prev);
315         for (j=0,ptrx=coordsx,ptry=coordsy;j<num_nodes;j++) {
316             ElleNodePlotXY(*(*nodes+j),&xy,&prev);
317             *ptrx = xy.x; ptrx++;
318             *ptry = xy.y; ptry++;
319             prev = xy;
320         }
321         area = polyArea(coordsx,coordsy,num_nodes);
322         free(coordsx);
323         free(coordsy);
324         //if (id) free(id);
325         return(area);
326     }
327
328     int intersectioncheck(int **child, int nchild)
329     {
330       int i, j, check, test, horizontal=0, wrap;
331       double dir, dir_test, l, l_test;
332       Coords start, end, temp;
333
334       // First get the node position of the start node which is the first one in the array
335       // Then get the position of the last node to determine the split direction against which
336       // all the other directions are tested.
337       //wrap=wrapcheck(&child, nchild);
338
339       ElleNodePosition(**child, &start);
340       ElleNodePlotXY(*(*child+(nchild-1)), &end, &start);
341       end.x = end.x - start.x;
```

```
342       end.y = end.y - start.y;
343       if (end.y==0)
344         horizontal=1;
345
346
347       // for the dir test the equation x2 = (x1/y1) * y2 is used
348
349       // if the split direction is horizontal y2 = (y1/x1) * x2 is used instead.
350
351       // this is the part between the brackets
352
353       if (horizontal==0)
354         dir=(end.x/end.y);
355       else if (horizontal==1)
356         dir=(end.y/end.x);
357
358       //determine the length of the split for the second part of the test (without root)
359       l=(end.y*end.y)+(end.x*end.x);
360
361       // get the node position of the 2nd node to determine whether this child
362       // is above or below the split.
363       ElleNodePlotXY(*(*child+1), &temp, &start);
364       //ElleNodePosition(*(*child+1), &temp);
365       temp.x = temp.x - start.x;
366       temp.y = temp.y - start.y;
367       // if there is no difference between the split direction and the test direction
368       // print out an error and quit.
369       if (temp.x==0 && temp.y==0) {
370         printf("Error: intersection check: first node check is the same as split direction\n");
371         return 0;
372       }
373       // otherwise determine the test direction
374
375       if (horizontal==0)
376         dir_test=dir*temp.y;
377       else if (horizontal==1)
378         dir_test=dir*temp.x;
379       /*  for x2 = (x1/y1) * y2 the part between the brackets has already been calculated
380        *  above. Now this party is multiplied with the y-part of the test location
381        *  to see whether this point is above or below the split boundary.
382        *
383        *  The result will be the virtual x location of the split boundary for the y value
384        *  of the test location. Afterwards the two locations are compared and depending
385        *  on if the result is larger or smaller then the test location a value is stored
386        *  that is needed for comparison of the other points afterwards.
387        *
388        *  If the virual location for every point is the same (in terms of smaller or larger) than the
389        *  test location there is no intersection.
390        */
391       if (temp.x>dir_test)
392         check=0;
393       else if (temp.x<dir_test)
394         check=1;
395       else
396         printf("ERROR: intersection check: check determination\n");
397
398       // this is needed for the second part of the test.
399       test=check;
400
401       // now the loop for all the other nodes starting from the 3rd to the penultimate node
402       for (i=2; i<(nchild-1) && test==check; i++) {
403         ElleNodePlotXY(*(*child+i), &temp, &start);
404         //ElleNodePosition(*(*child+i), &temp);
405         temp.x = temp.x - start.x;
406         temp.y = temp.y - start.y;
407
408         if (horizontal==0)
409           dir_test=dir*temp.y;
410         else if (horizontal==1)
411           dir_test=dir*temp.x;
412
413         if (temp.x>dir_test)
414           test=0;
415         else if (temp.x<dir_test)
416           test=1;
417         else if (temp.x==dir_test)
418           test=2;
419         else
420           printf("ERROR: intersection check: check determination 2\n");
421
422         // Second part of the test.
423         // If the check and the test differs, the length of the split against the check is tested
424         // If the test length is longer than the split length then the split is still ok -- no intersection.
425         if (test!=check) {
426           if ((l_test=(temp.y*temp.y)+(temp.x*temp.x))<l) {
427             //printf("Error: intersection check: length: INTERSECTION\n");
428             break;
429           }
430           else if (l_test==l) {
431             //printf("Error: intersection check: length: possible Intersection\n");
432             break;
433           }
434           else if (l_test>l)
435             // test is set equal to check again, because split is possible if
436             // l_test is longer than l
437             test=check;
438           else
439             printf("Error: intersection check: length: undefined error\n");
440         }
441
442       }
```

```
443       // if test was equal to check for the whole loop, return that the split is possible without intersection.
444       if (test==check)
445         return 1;
446       else
447         return 0;
448     }
449
450     int flynnsplit2(int flynnindex, int start, int end, int **child1, int **child2, int *nchild1, int *nchild2, int **c1, int
451     **c2)
452     {
453       int i, j;
454       ERegion rgn1, rgn2;
455
456          // create 2 new children
457          // search for 2 spare flynns and set them as childs of the parent grain
458       // assign an ERegion to them.
459          **c1 = ElleFindSpareFlynn(); // first -> end
460          **c2 = ElleFindSpareFlynn(); // end -> first
461          ElleAddFlynnChild(flynnindex, **c1);
462          ElleAddFlynnChild(flynnindex, **c2);
463          rgn1 = **c1;
464          rgn2 = **c2;
465
466          // First Child
467          // for all the nodes in the array of child 1
468          for (i=0;i<*nchild1;i++) {
469           // for the last node, connect it with the first node
470           if (i==*nchild1-1) {
471             // find the NO_NB entry in the neighbours of the last node
472             j = ElleFindNbIndex(*(*child1),*(*child1+i));
473             // assign this NO_NB entry in the neighbours of the last node to the first node --> new triple node
474           ElleSetNeighbour(*(*child1+i), j, *(*child1), &rgn1);
475           //printf("End of Child1: %d - %d: %d\n", *(*child1+i), *(*child1), rgn1);
476           } else {
477             // find out which of the 3 neighbours of a node the next node is
478           j = ElleFindNbIndex(*(*child1+(i+1)),*(*child1+i));
479           // set the next node as this neighbour for the node
480           ElleSetNeighbour(*(*child1+i), j, *(*child1+(i+1)), &rgn1);
481           //ElleSetRegionEntry(*child1+i,j,rgn1);
482           //printf("child1: %d - %d: %d\n", *(*child1+i), *(*child1+(i+1)), rgn1);
483           }
484          }
485          // set the first node as first node of child 1
486          ElleSetFlynnFirstNode(**c1, start);
487
488          // Second Child
489          // do the same for child 2 except that start and end nodes are exchanged.
490          for (i=0;i<*nchild2;i++) {
491           if (i==*nchild2-1) {
492             j = ElleFindNbIndex(*(*child2),*(*child2+i));
493           ElleSetNeighbour(*(*child2+i), j, *(*child2), &rgn2);
494           //printf("End of Child2\n");
495           } else {
496           j = ElleFindNbIndex(*(*child2+(i+1)),*(*child2+i));
497           ElleSetNeighbour(*(*child2+i), j, *(*child2+(i+1)), &rgn2);
498           //ElleSetRegionEntry(*child2+i,j,rgn2);
499           //printf("child2\n");
500           }
501       }
502          // set the end node as first node of child 1
503          ElleSetFlynnFirstNode(**c2, end);
504          // add new double nodes to the newly created boundary.
505          ElleAddDoubles();
506     }
507
508     // not from me, a common quicksort alogarithm
509     void sortstruct(DEVIATION items[], int left, int right)
510     {
511
512       register int i, j;
513       double x;
514       DEVIATION temp;
515
516       i = left; j = right;
517       x = items[(left+right)/2].error;
518
519       do {
520         while(items[i].error < x && (i < right)) i++;
521         while(items[j].error > x && (j > left)) j--;
522         if(i <= j) {
523           temp = items[i];
524           items[i] = items[j];
525           items[j] = temp;
526           i++; j--;
527         }
528       } while(i <= j);
529
530       if(left < j) sortstruct(items, left, j);
531       if(i < right) sortstruct(items, i, right);
532     }
533
534     int assignstruct(int **id, double dir, int num_nodes, int *possis)
535     {
536       int i, j, k=0;
537       double dir_test;
538       Coords n1, n2;
539
540       // count the maximum connections that need to be tested num_nodes!
541       // without the triple nodes because they aren't used
542       for (j=0,i=0; j<num_nodes; j++)
543         if (ElleNodeIsDouble(*(*id+j)))
```

```
544         i++;
545
546       // only do that if there are at least MINDNODES double nodes.
547       if (i>=MINDNODES) {
548         // substract 1 from the amount of double nodes because that one is used as starting
549         // node and therefore isn't used in this calculation. Then calculate factorial of the nodes
550         for (--i,j=0;i>0;i--)
551           j+=i;
552
553         //printf("Possis: %d\n", j);
554
555
556         if ((dev = (DEVIATION *)malloc(j * sizeof *dev))==0) {
557           printf("ERROR: assignstruct: Malloc_Err: deviation struct\n");
558           return 0;
559         }
560
561         //find a pair of nodes fitting the direction
562         /* compares the direction from every node to every other node
563          * only accepts double nodes as possible split nodes
564          */
565         for(i=0, k=0; i<num_nodes; i++) {
566           if (ElleNodeIsDouble(*(*id+i))) {
567             ElleNodePosition(*(*id+i), &n1);
568             for(j=i; j<num_nodes ;j++) {
569               if (i!=j && ElleNodeIsDouble(*(*id+j))) {
570                 //ElleNodePosition(*(*id+j), &n2);
571                 ElleNodePlotXY(*(*id+j), &n2, &n1);
572                 dir_test=atan((n2.y-n1.y)/(n2.x-n1.x));
573                 dev[k].x = *(*id+i);
574                 dev[k].y = *(*id+j);
575                 dev[k++].error = fabs(dir-dir_test);
576               }
577             }
578           }
579         }
580
581         *possis=k;
582         return 1;
583       } else {
584         //printf("ERROR: assignsstruct: too less d-nodes: no possibility to split\n");
585         return 2;
586       }
587     }
588
```

2 GROWTH + SPLIT

```
#include <vector>
#include <algorithm>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include "attrib.h"
#include "nodes.h"
#include "file.h"
#include "display.h"
#include "check.h"
#include "error.h"
#include "runopts.h"
#include "init.h"
#include "general.h"
#include "stats.h"
#include "update.h"
#include "interface.h"
#include "polygon.h"
#include "../split2/split2.elle.cc"

#define PI 3.141592653589793
#define DtoR PI/180
#define RtoD 180/PI

using std::vector;

int InitGG_Split(void);
int Init_GG_Split(void);
int GG_Split(int splitmode, double chance, double min_area, double max_area, double mcs, int x);
int MoveDoubleJ(int node1);
int MoveTripleJ(int node1);
extern void GetRay(int node1,int node2,int node3,double *ray,Coords *movedist);
void MoveFlynnNodes(int **nodes, int num, int moves);
void TimeWrite(FILE **where);
double ListBNodes(int **nodes, int n, double *dir_x, double *dir_y);

extern runtime_opts Settings_run;

/* mid_area: the average of all grain areas
 * TotalTime: not really used
 * max_split_age: age a daughter grain has to be before it can split again
 * chance: chance for a grain to split
 * min_child_area: minimum fraktion of the area of the parent flynn for the 2 daughter flynns
 * max_area: area when the chance comes to 100% for splitting
 */

double TotalTime, gb_energy;
FILE *fp;  //this is where the log is written.

int InitGG_Split(void)
{
    int err=0;
    char *infile;

    printf("Usage:\ncommand line parameter -u x1 x2 x3 (x4) (x5) (x6) -- (x)=optional, *=standard (used if nothing else is
supplied\nx1: splitmode\n\t1* = every grain same chance (x2)\n\t2 every grain starting from min_area (x5) same chance
(x2)\n\t3 increasing chance from min_area (x5) with chance (x2) to max_area (x6) with 100%% chance\n");
  printf("x2: split chance from 0 to 1*\nx3: randomshuffle&randomD forward - supplied int-1, 0* is default\nx4: restart step -
- supplied in case of crash to restart at the given step number (e.g. for numeration), 0* is default\nx5: min_area - double
(should be supplied in split mode 2&3)\nx6: max_area - double (should be supplied in split mode 3)\n");


    ElleReinit();
    ElleSetRunFunction(Init_GG_Split);

    infile = ElleFile();


    if (strlen(infile)>0) {
        if (err=ElleReadData(infile)) OnError(infile,err);

        ElleAddDoubles();
    }

  if (!ElleFlynnAttributeActive(SPLIT))
      ElleInitFlynnAttribute(SPLIT);
}

int Init_GG_Split(void)
{
  int split_mode, start_stage, x, i;
  double chance, min_area, max_area, min_child_area;

  UserData userdata;

  ElleUserData(userdata);
    // 1=every grain has same chance2split,
    // 2=every grain>MinArea same chance2split,
    // 3= grains<MinArea 0%chance till grains>MaxArea 100%chance2split,
  split_mode = (int)userdata[0];
    if (split_mode<1 || split_mode>3)
     split_mode=1;
    chance = (double)userdata[1];
```

```
95        if (chance<0)
96         chance=0;
97        else if (chance>1)
98         chance=1;
99        x = (int)userdata[2];
100       x = fabs(x);
101       x -= 1;
102       for (i=0; i<x; i++)
103      ElleRandomD();
104       min_area = (double)userdata[4];
105       max_area = (double)userdata[5]; // area where chance to split becomes 100% used for split mode 3
106       if (min_area>max_area) {
107        printf("Error: min_area > max_area: correction of max_area\n");
108        max_area=min_area*.5;
109       }
110     // last stage if simulations has crashed.
111       if ((start_stage=(int)userdata[3])!=1)
112        Settings_run.Count = start_stage;
113
114       min_child_area = .4;
115       // command line can be started with e.g. -u 0.01 0.001 0.35
116
117       //printf("How many steps should the randomD generator be forwarded? - Enter an int.\n");
118     //scanf("%d", &x);
119     //x = fabs(x);
120
121
122       fp=fopen("log.txt", "a");
123       //filename for the log
124
125       fprintf(fp,"\n**********************************\n************* RUN
126    ***************\n**********************************\n\n");
127       TimeWrite(&fp);
128       fprintf(fp, "%d\tforward of randomD\n", x);
129       fprintf(fp, "%d\t\tsplit mode\n", split_mode);
130       fprintf(fp, "%E\tmin grain size\n", min_area);
131       fprintf(fp, "%E\tmax grain size\n", max_area);
132     fprintf(fp, "%E\tchance for a grain to split\n", chance);
133     fprintf(fp, "%E\tminimum area for child grains as fraction of the parent grain\n", min_child_area);
134     fflush(fp);
135
136     fclose(fp);
137
138     TotalTime=0.0;
139
140     GG_Split(split_mode, chance, min_area, max_area, min_child_area, x);
141   }
142
143   void TimeWrite(FILE **where)
144   {
145     char timestore[80];
146
147     time_t rawtime;
148     struct tm * timeinfo;
149
150     time ( &rawtime );
151     timeinfo = localtime ( &rawtime );
152     // for the time...
153
154     strftime(timestore, 80, "Date: %d/%m/%Y, %H:%M:%S\n", timeinfo);
155     fprintf(*where,"%s", timestore);
156   }
157
158   int GG_Split(int splitmode, double chance, double min_area, double max_area, double mcs, int x)
159   {
160
161     int i=0, j=0, n=0, c1, c2, k, savestep;
162       int maxn, maxf;
163       int errors=0, splits=0, small_errors=0, nd=0, *nodes, num;
164       vector<int> seq;
165       double a, test, try_x, try_y;
166
167       savestep = Settings_run.save.frequency;
168
169       fp=fopen("log.txt", "a");
170
171       fprintf(fp,"**************START***************\nStages: %d\n**********************************\n", EllemaxStages());
172       fprintf(fp,"STEP\tSplits+ndSplits\tsGrains\tErrors\n");
173
174       if (ElleCount()==0) ElleAddDoubles();
175       if (ElleDisplay()) EllePlotRegions(ElleCount());
176     ElleCheckFiles();
177
178     gb_energy = (1e-7)*ElleTimestep();
179
180
181     for (i=0;i<EllemaxStages();i++) {
182
183           maxn = ElleMaxNodes();
184           seq.clear();
185           for (j=0;j<maxn;j++)
186             if (ElleNodeIsActive(j))
187               seq.push_back(j);
188
189           random_shuffle(seq.begin(),seq.end());
190           maxn = seq.size();
191           for (n=0;n<maxn;n++) {
192               j=seq[n];
193               if (ElleNodeIsActive(j)) {
194                   if (ElleNodeIsDouble(j)) {
195                       MoveDoubleJ(j);
```

```
                    ElleCheckDoubleJ(j);
                }
                else if (ElleNodeIsTriple(j)) {
                    MoveTripleJ(j);
                    ElleCheckTripleJ(j);
                }
            }
        }
    seq.clear();
    maxf = ElleMaxFlynns();

    for (j=0;j<maxf;j++)
      if (ElleFlynnIsActive(j))
        seq.push_back(j);
    for (j=0;j<x;j++)
      random_shuffle(seq.begin(),seq.end());
    maxf = seq.size();

    for (n=0;n<maxf;n++) {
      j=seq[n];

      //area of the flynn
      a = fabs(ElleRegionArea(j));
      test = ElleRandomD();
      if (splitmode == 3)
        chance += ((a-min_area)/(max_area-min_area));
      //printf("Chance: %f\n", chance);
      if (splitmode == 1) {
        if (test<chance) {
          ElleFlynnNodes(j, &nodes, &num);
          ListBNodes(&nodes, num, &try_x, &try_y);
          k = directionsplit2(j, try_x, try_y, mcs, &c1, &c2);
          if (k==1) {
            splits++;
            EllePromoteFlynn(c1);
            EllePromoteFlynn(c2);
            ElleRemoveFlynn(j);
          }
          else if (k==2)
            small_errors++;
          else if (k==3) {
            nd++;
            EllePromoteFlynn(c1);
            EllePromoteFlynn(c2);
            ElleRemoveFlynn(j);
          }
          else
            errors++;
        }
      }
      else if (splitmode == 2 || splitmode ==3) {
        if (a >= min_area) {
          if (test<chance) {
            ElleFlynnNodes(j, &nodes, &num);
            ListBNodes(&nodes, num, &try_x, &try_y);
            k = directionsplit2(j, try_x, try_y, mcs, &c1, &c2);
            if (k==1) {
              splits++;
              EllePromoteFlynn(c1);
              EllePromoteFlynn(c2);
              ElleRemoveFlynn(j);
              ElleSetFlynnIntAttribute(c1,1,SPLIT);
              ElleSetFlynnIntAttribute(c2,1,SPLIT);
            }
            else if (k==2)
              small_errors++;
            else if (k==3) {
              nd++;
              EllePromoteFlynn(c1);
              EllePromoteFlynn(c2);
              ElleRemoveFlynn(j);
              ElleSetFlynnIntAttribute(c1,1,SPLIT);
              ElleSetFlynnIntAttribute(c2,1,SPLIT);
            }
            else
              errors++;
          }
        }

      }
    }

    if (savestep == 0) {
      if (i%5000==0) //Write the time in the log every 10k steps
        TimeWrite(&fp);
      if (i%200==0)
        fprintf(fp,"%d\t%d+%d\t%d\t%d\n", Settings_run.Count, splits, nd, small_errors, errors);
      fflush(fp);
    } else {
      if (i%(savestep*5)==0) //Write the time in the log every 10k steps
        TimeWrite(&fp);
      if (i%savestep==0)
        fprintf(fp,"%d\t%d+%d\t%d\t%d\n", Settings_run.Count, splits, nd, small_errors, errors);
      fflush(fp);
    }

        ElleUpdate();
    }
  fprintf(fp, "%d\t%d+%d\t%d\t%d\nEND: STEP\tSplits+ndSplits\tsGrains\tErrors\n", Settings_run.Count, splits, nd,
small_errors, errors);
    fclose(fp);
```

```
297     }
298
299     double ListBNodes(int **nodes, int n, double *dir_x, double *dir_y)
300     {
301       int i, j;
302       double l=0, cl=0;
303       Coords node, dist;
304
305       // Checks distance from every node to the other nodes and keeps the direction perpendicular to the longest direction
306
307       for (i=0; i<n; i++) {
308         ElleNodePosition(*(*nodes+i), &node);
309         for (j=0; j<n; j++) {
310           ElleNodePlotXY(*(*nodes+j),&dist,&node);
311           dist.x-=node.x;
312           dist.y-=node.y;
313           cl = sqrt((double)((dist.x*dist.x)+(dist.y*dist.y)));
314           if (l<cl) {
315             l=cl;
316             *dir_x = -dist.y/cl;
317             *dir_y = dist.x/cl;
318           }
319         }
320
321
322       }
323       return l;
324     }
325
326     void MoveFlynnNodes(int **nodes, int num, int moves)
327     {
328       int i, n, j, maxn;
329       vector<int> seq;
330     /*
331       printf("%d:", flynn);
332       for (j=0;j<num;j++)
333         printf(" %d", *(*nodes+j));
334       printf("\n");
335     */
336       for (j=0;j<num;j++)
337         if (ElleNodeIsActive(*(*nodes+j)))
338           seq.push_back(*(*nodes+j));
339       random_shuffle(seq.begin(),seq.end());
340       maxn = seq.size();
341       //printf("%d\n", maxn);
342       for (i=0;i<moves;i++) {
343         for (n=0;n<maxn;n++) {
344           j=seq[n];
345           if (ElleNodeIsActive(j)) {
346             if (ElleNodeIsDouble(j)) {
347               MoveDoubleJ(j);
348               ElleCheckDoubleJ(j);
349             }
350             else if (ElleNodeIsTriple(j)) {
351               MoveTripleJ(j);
352               ElleCheckTripleJ(j);
353             }
354           }
355         }
356       }
357     }
358
359     int MoveDoubleJ(int node1)
360     {
361         int i, nghbr[2], nbnodes[3], err;
362         double maxV,ray,deltaT,vlen;
363         double switchDist, speedUp;
364         Coords xy1, movedist;
365
366         switchDist = ElleSwitchdistance();
367         speedUp = ElleSpeedup() * switchDist * switchDist * 0.02;
368         maxV = ElleSwitchdistance()/5.0;
369         /*
370          * allows speedUp to be 1 in input file
371          */
372         //gb_energy = speedUp;
373         deltaT = 0.0;
374         /*
375          * find the node numbers of the neighbours
376          */
377         if (err=ElleNeighbourNodes(node1,nbnodes))
378             OnError("MoveDoubleJ",err);
379         i=0;
380         while (i<3 && nbnodes[i]==NO_NB) i++;
381         nghbr[0] = nbnodes[i]; i++;
382         while (i<3 && nbnodes[i]==NO_NB) i++;
383         nghbr[1] = nbnodes[i];
384
385         GetRay(node1,nghbr[0],nghbr[1],&ray,&movedist);
386         if (ray > 0.0) {
387         /*if (ray > ElleSwitchdistance()/100.0) {*/
388             vlen = gb_energy/ray;
389             if (vlen > maxV) {
390                 vlen = maxV;
391                 deltaT = 1.0;
392             }
393             if (vlen>0.0) {
394                 movedist.x *= vlen;
395                 movedist.y *= vlen;
396             }
397             else {
```

```
398                 movedist.x = 0.0;
399                 movedist.y = 0.0;
400             }
401             TotalTime += deltaT;
402             ElleUpdatePosition(node1,&movedist);
403         }
404         else {
405             vlen = 0.0;
406         }
407     }
408
409     int MoveTripleJ(int node1)
410     {
411         int i, nghbr[3], finished=0, err=0;
412         double maxV,/*gb_energy[3],*/ray[3],deltaT,vlen[3],vlenTriple;
413         double switchDist, speedUp;
414         Coords xy1, movedist[3], movedistTriple;
415
416         switchDist = ElleSwitchdistance();
417         /*
418          * allows speedUp to be 1 in input file
419          */
420         speedUp = ElleSpeedup() * switchDist * switchDist * 0.02;
421         maxV = switchDist/5.0;
422         //for (i=0;i<3;i++) gb_energy[i] = speedUp;
423         deltaT = 0.0;
424         /*
425          * find the node numbers of the neighbours
426          */
427         if (err=ElleNeighbourNodes(node1,nghbr))
428             OnError("MoveTripleJ",err);
429
430         GetRay(node1,nghbr[0],nghbr[1],&ray[0],&movedist[0]);
431         GetRay(node1,nghbr[1],nghbr[2],&ray[1],&movedist[1]);
432         GetRay(node1,nghbr[2],nghbr[0],&ray[2],&movedist[2]);
433         for(i=0;i<3;i++) {
434             if (ray[i] > 0.0) {
435             /*if (ray[i] > switchDist/100.0) {*/
436                 vlen[i] = gb_energy/ray[i];
437                 /*if (vlen[i] > maxV) vlen[i] = maxV;*/
438             }
439             else {
440                 vlen[i] = 0.0;
441                 finished = 1;
442             }
443         }
444         if (!finished) {
445             for(i=0;i<3;i++) {
446                 if (vlen[i] < maxV) {
447                     movedist[i].x *= vlen[i];
448                     movedist[i].y *= vlen[i];
449                 }
450                 else {
451                     movedist[i].x *= maxV;
452                     movedist[i].y *= maxV;
453                 }
454             }
455             movedistTriple.x = movedist[0].x+movedist[1].x+movedist[2].x;
456             movedistTriple.y = movedist[0].y+movedist[1].y+movedist[2].y;
457             vlenTriple = sqrt(movedistTriple.x*movedistTriple.x +
458                               movedistTriple.y*movedistTriple.y);
459             if (vlenTriple > maxV) {
460                 vlenTriple = maxV/vlenTriple;
461                 movedistTriple.x *= vlenTriple;
462                 movedistTriple.y *= vlenTriple;
463                 deltaT = 1.0;
464             }
465             if (vlenTriple <= 0.0) movedistTriple.x = movedistTriple.y = 0.0;
466
467             TotalTime += deltaT;
468         }
469         else {
470             ElleNodePosition(node1,&xy1);
471             ElleNodePlotXY(nghbr[0],&movedist[0],&xy1);
472             ElleNodePlotXY(nghbr[1],&movedist[1],&xy1);
473             ElleNodePlotXY(nghbr[2],&movedist[2],&xy1);
474             for(i=0;i<3;i++) {
475                 movedist[i].x = movedist[i].x - xy1.x;
476                 movedist[i].y = movedist[i].y - xy1.y;
477             }
478             movedistTriple.x = (movedist[0].x+movedist[1].x+movedist[2].x)/2.0;
479             movedistTriple.y = (movedist[0].y+movedist[1].y+movedist[2].y)/2.0;
480     #if XY
481             vlenTriple = sqrt(movedistTriple.x*movedistTriple.x +
482                               movedistTriple.y*movedistTriple.y);
483             if (vlenTriple > maxV) {
484                 vlenTriple = maxV/vlenTriple;
485                 movedistTriple.x *= vlenTriple;
486                 movedistTriple.y *= vlenTriple;
487                 deltaT = 1.0;
488             }
489     #endif
490         }
491         ElleUpdatePosition(node1,&movedistTriple);
492     }
493
```

3 POLY PHASE GRAIN BOUNDARY MIGRATION

3.1 HEADER: GBM_PP_UNODES.H

```
#ifndef _gbm_pp_elle_h
#define  _gbm_pp_elle_h
#include <cstdio>
#include <cmath>
#include <cstring>
#include <vector>
#include <algorithm>
#include <list>
#include <iostream>
#include <fstream>
#include <sstream>
#include "attrib.h"
#include "nodes.h"
#include "unodes.h"
#include "file.h"
#include "display.h"
#include "check.h"
#include "error.h"
#include "runopts.h"
#include "init.h"
#include "general.h"
#include "stats.h"
#include "update.h"
#include "interface.h"
#include "crossings.h"
#include "mineraldb.h"
#include "polygon.h"
#include "movenode_unodes.h"
//#include "growthstats.h"
/*#define PI 3.1415926
#define DTOR PI/180
#define RTOD 180/PI*/


typedef struct
{
  int flynn;
  int phase;
} Flynnies;

typedef struct
{
  double mobility;
  double b_energy;
  double dGbActEn;
} PhaseBoundaryProps;

typedef struct
{
  int infinite_diff;
  int cluster_diff;
  int diffusion_times;
  double elasticity;
  double scale;
  double kappa;
  int merge;
} PhaseProps;

typedef struct
{
  int no_phases;
  int p_en;
  int p_track;
  PhaseProps phasep[15];
  PhaseBoundaryProps pairs[120][120];
} AllPhases;

typedef struct
{
  int node;
  int p;
  int nb1;
  int nb2;
  int nb1_p1;
  int nb1_p2;
  int nb2_p1;
  int nb2_p2;
  int not_diff;
  double newconc;
} DiffNodes;

class clusters
{
public:
  clusters( std::vector<int>, double );
  ~clusters ();
```

```
 87       std::vector<int> ReturnClusterFlynns ( void );
 88       double  ReturnClusterArea ( void );
 89     private:
 90       std::vector<int> vFlynns;
 91       double dArea;
 92     };
 93
 94     class clusterTracking
 95     {
 96     public:
 97       clusterTracking();
 98       ~clusterTracking();
 99       bool writeInitialData( const char* );
100       bool writeData( const char*, int );
101       void setClusterAreas( void );
102       void findSplit( void );
103       void findMerge( void );
104       void findClusters( void );
105       void updateClusters( void );
106       void checkDoubleClusterAreaLoop( void );
107       double returnClusterAreaEnergy ( int , Coords * );
108
109     private:
110       int iMaxFlynns; // max Flynns. Has to by initialized with this value
111       double dAreaShift, dMultiplierA, dMultiplierB, dMultiplierC, dMultiplierD;
112       //std::vector<clusters> vClusters;
113       std::vector<int> vFlynns, vFlynnPhase;
114       std::vector<double> vPhaseAreas; // Areas of the phases
115       std::vector<std::vector<double> > vPhasesClusterAreas; // Areas of the Clusters for all Clusterphases
116
117       // vClusterPhases -> stores Phases which are set to clusterdiffusion
118       // |---------|
119       // | Phase 1 | --> Phase 1 is set to cluster diffusion
120       // |---------|     -----------------------------
121       // |   ...   |     | Cluster 1 | Cluster 2 | ...
122       //                 -----------------------------
123       //                   --> First Cluster of Flynns of Phase 1
124       //                       |---------|
125       //                       | Flynn 1 | --> First Flynn of that Cluster
126       //                       |---------|
127       //                       | Flynn 2 |
128       //                       |---------|
129       //                       |   ...   |
130       std::vector<std::vector<std::vector<int> > > vPhasesClusters; // Flynns of all Clusters of all Clusterphases
131
132       std::vector<int> vClusterPhases; // a copy of the ClusterDiffPhases (copied in the Constructor)
133
134
135       // FUNKTIONEN
136
137       std::vector<double> returnMultiplier ( std::vector<double > );
138       double returnFlynnAreaChange ( int, int, Coords * );
139       std::vector<double> returnClusterAreaChange ( std::vector<std::vector<int > > , int, Coords * );
140       bool checkDoubleClusterArea( int, int, int );
141       void getPhaseAreas( void );
142       void getClusters( void );
143       void getClusterAreas( void );
144       void resolveSplit( std::vector<std::vector<int > > );
145       void resolveMerge( int, int, std::list<double> );
146       bool resolveDoubleClusterArea( int, int, int, double );
147     };
148
149
150
151
152     int InitGrowth( void );
153     int GBMGrowth( void );
154
155     double GetNodeEnergy( int, Coords *, clusterTracking * );
156     int GGMoveNode( int, Coords * );
157
158     int Read2PhaseDb(char *dbfile, AllPhases *phases);
159     double CheckPair(int node1, int node2, int type);
160
161     int StoreAreaChange(int node, Coords *vector, int node_type);
162     int GetArea(int node, double area[], double phase_area[], int type_i[], int nodetype, Coords *loc);
163     double ReturnAreaEnergy(int node, Coords *location);
164     double ReturnArea(ERegion poly, int node, Coords *pos);
165
166     bool fileExists( const char* );
167
168     int diffusearea(int mode, int max);
169     int shiftarea(int node, int n1, int n2, int i);
170     int diffuse_dn(int position, DiffNodes *nodes);
171     void writenewconc(int number, DiffNodes *nodes);
172     void mergeair(int mode);
173     int savearea(int mode, int max);
174     int setupflynnies(void);
175
176     int UnodePhaseUpdate();
177     int UnodePhaseShift();
178     int AssignUnodeProperties(); // Enriques melt tracking function
179     #endif
```

3.3 CODE: GBM_PP_UNODES.CC

```
#include "gbm_pp_unodes.elle.h"

using namespace std;

extern runtime_opts Settings_run;

// Phasenspeicherplaetze
int iClusterNodeCount = N_ATTRIB_A; //not used atm
int attrib[2] = { N_ATTRIB_B, N_ATTRIB_C };
int iFlynnPhase = F_ATTRIB_A;
int iFlynnCluster = F_ATTRIB_C;
int iUnodePhase = U_ATTRIB_C;
int iUnodeConc = U_ATTRIB_A;
//
int iUnodeUpdateMethod = 1;
// 1 = Find Unodes in a Flynn and update them accordingly with the FlynnPhase in iUnodePhase.
// 0 = Find according Flynn for each Unode and update them with the FlynnPhase  in iUnodePhase.
double dCheckEnergy[4];
int bCheckEnergy = 0;
int iCheckEnergy = 0;
int iMinTjs = 0;

#define MAX_PHASES 2
#define MAX_DIFF_STEPS 1000 //the maximum number of diffusion steps --> overwrites the values given in the config file.
#define MIN_DIFF_DT 500 // something that is used in the gbdiff code. for the moment I just copied it.

AllPhases phases;
UserData userdata;
Flynnies *grains;

// the phases which have to diffuse by different kinds...
// used by a few different processes.
list<int> lInfDiffPhases, lClustDiffPhases, lFicksDiffPhases, lAllPhases;

FILE *fp;

//The nodes are moved along the energy gradient
//Written by Dr. J.K. Becker
//Modified by Jens Roessiger

/*!/brief Calculates the (Surface) Energy of a node

This really only calculates the surface energy of the node, nothing else. General equation is E=en(l1+l2+l3)*lengthscale
with E=energy, en=surface energy and l1/l2/l3 the length of the segments next to the node adjusted to the lengthscale */
double GetNodeEnergy( int node, Coords * xy )
{
    int err, n, node2, node1, node3, nbnode[3], mineral, rgn[3];
    Coords n1, n2, n3, v1, v2, v3;
    double l1, l2, l3, E, en = 0;
    double bodyenergy=0, energyofsurface=0;
    //Get the neighbouring nodes
    if ( err = ElleNeighbourNodes( node, nbnode ) )
        OnError( "MoveNode", err );
    n = 0;
    //and put them into variables. In case of a double node, one is NO_NB and we don't want to use
    //that
    while ( n < 3 && nbnode[n] == NO_NB )
        n++;
    node1 = nbnode[n];
    n++;
    while ( n < 3 && nbnode[n] == NO_NB )
        n++;
    node2 = nbnode[n];
    n = 0;
    //see if the neighbouring nodes are active. Do we need that? Don't think so...
    if ( ElleNodeIsActive( node1 ) )
        n++;
    if ( ElleNodeIsActive( node2 ) )
        n++;
    //Get positions of neighbouring nodes
    ElleNodePlotXY( node1, & n1, xy );
    ElleNodePlotXY( node2, & n2, xy );
    //we don't really need the positions, we just need the length of the segments
    v1.x = n1.x - xy->x;
    v1.y = n1.y - xy->y;
    v2.x = n2.x - xy->x;
    v2.y = n2.y - xy->y;
    l1 = GetVectorLength( v1 );
    l2 = GetVectorLength( v2 );
    //if the node is a triple, we have to get the third node and the length of the segment
    if ( ElleNodeIsTriple( node ) )
    {
        node3 = nbnode[2];
        ElleNodePlotXY( node3, & n3, xy );
        v3.x = n3.x - xy->x;
        v3.y = n3.y - xy->y;
        l3 = GetVectorLength( v3 );
    }
  // Check which combination of phases is present at the current bounary segment and return the energy of that.
  E = l1*CheckPair(node, node1, 1);
  E += l2*CheckPair(node, node2, 1);
  if (ElleNodeIsTriple(node))
    E += l3*CheckPair(node, node3, 1);
  E *= ElleUnitLength();
  // printf("%le %le %le %le\n",l1,l2,l3,E);
     return E;
```

```
 97  }
 98
 99  int InitGrowth()
100  {
101      int err = 0;
102      char * infile;
103      char *dbfile;
104  //    unsigned char found = 0;
105
106
107      ElleReinit();
108      ElleSetRunFunction( GBMGrowth );
109
110      //if (!ElleNodeAttributeActive(ATTRIB_A))
111        //ElleInitNodeAttribute(ATTRIB_A);
112      //if (!ElleNodeAttributeActive(ATTRIB_B))
113        //ElleInitNodeAttribute(ATTRIB_B);
114      //if (!ElleNodeAttributeActive(ATTRIB_C))
115        //ElleInitNodeAttribute(ATTRIB_C);
116
117  // ElleSetDefaultNodeAttribute(0, ATTRIB_A);
118  // ElleSetDefaultNodeAttribute(0, ATTRIB_B);
119  // ElleSetDefaultNodeAttribute(0, ATTRIB_C);
120  /*
121    if (!ElleFlynnAttributeActive(F_ATTRIB_A)) {
122      ElleInitFlynnAttribute(F_ATTRIB_A);
123      cout << "NO PHASES HAVE BEEN SET!" << endl << "This is a single phase process..." << endl;
124    }
125    if (!ElleFlynnAttributeActive(F_ATTRIB_B))
126      ElleInitFlynnAttribute(F_ATTRIB_B);
127
128  // ElleSetDefaultFlynnAttribute(-1, F_ATTRIB_A);
129  // ElleSetDefaultFlynnAttribute(-1, F_ATTRIB_B);
130
131  */
132    ElleUserData(userdata);
133    ElleSetOptNames("Starting TimeStep","unused","unused","unused","unused","unused","unused","unused","unused");
134
135
136
137      infile = ElleFile();
138      if ( strlen( infile ) > 0 )
139      {
140          if ( err = ElleReadData( infile ) )
141              OnError( infile, err );
142          ElleAddDoubles();
143      }
144
145      if((dbfile=ElleExtraFile())<0)
146        OnError( dbfile, 0 );
147
148      if(!Read2PhaseDb(dbfile, &phases)) //dbfile
149       OnError( dbfile, 0 );
150
151      if ( phases.p_track >= 0 ) {
152     if(!ElleUnodeAttributeActive(U_ATTRIB_A))
153       ElleInitUnodeAttribute(U_ATTRIB_A);
154     if(!ElleUnodeAttributeActive(U_ATTRIB_B))
155       ElleInitUnodeAttribute(U_ATTRIB_B);
156     if(!ElleUnodeAttributeActive(U_ATTRIB_C))
157       ElleInitUnodeAttribute(U_ATTRIB_C);
158     ElleSetDefaultUnodeAttribute(0, U_ATTRIB_A);
159     ElleSetDefaultUnodeAttribute(0, U_ATTRIB_B);
160     ElleSetDefaultUnodeAttribute(-1.0, U_ATTRIB_C);
161    }
162    else if ( phases.p_track == -1 ) {
163     if( !ElleUnodeAttributeActive( iUnodePhase ) )
164       ElleInitUnodeAttribute( iUnodePhase );
165     ElleSetDefaultUnodeAttribute( 0, iUnodePhase );
166    }
167
168
169  }
170
171  /* !/brief this is called to when it is time to move a node (and also to check if a node has to be moved) */
172  int GGMoveNode( int node, Coords *xy, clusterTracking *clusterData)
173  {
174      double e[4], a[4], ca[4], switchd = ElleSwitchdistance()/100; //added a for the area
175      Coords oldxy, newxy, prev;
176
177      ElleNodePosition( node, &oldxy );
178      ElleNodePrevPosition( node, &prev );
179      newxy.x = oldxy.x + switchd;
180      newxy.y = oldxy.y;
181      e[0] = GetNodeEnergy( node, &newxy );
182      a[0] = 0; //ReturnAreaEnergy( node, &newxy );
183      ca[0] = clusterData->returnClusterAreaEnergy( node, &newxy );
184      newxy.x = oldxy.x - switchd;
185      e[1] = GetNodeEnergy( node, &newxy );
186      a[1] = 0; //ReturnAreaEnergy( node,&newxy );
187      ca[1] = clusterData->returnClusterAreaEnergy( node, &newxy );
188      newxy.x = oldxy.x;
189      newxy.y = oldxy.y + switchd;
190      e[2] = GetNodeEnergy( node, &newxy );
191      a[2] = 0; //ReturnAreaEnergy( node, &newxy );
192      ca[2] = clusterData->returnClusterAreaEnergy( node, &newxy );
193      newxy.y = oldxy.y - switchd;
194      e[3] = GetNodeEnergy( node, &newxy );
195      a[3] = 0; //ReturnAreaEnergy( node, &newxy );
196      ca[3] = clusterData->returnClusterAreaEnergy( node, &newxy );
197      if ( bCheckEnergy == 0 ) {
```

```
198       dCheckEnergy[0] = 0;
199       dCheckEnergy[1] = 0;
200       dCheckEnergy[2] = 0;
201       dCheckEnergy[3] = 0;
202       bCheckEnergy = 1;
203     }
204     if ( bCheckEnergy == 1 ) {
205       dCheckEnergy[0] += ( ( e[0] - e[1] ) / ( 2 * switchd ) );
206       dCheckEnergy[1] += ( ( e[2] - e[3] ) / ( 2 * switchd ) );
207       dCheckEnergy[2] += ( ( ca[0] - ca[1] ) / ( 2 * switchd ) );
208       dCheckEnergy[3] += ( ( ca[2] - ca[3] ) / ( 2 * switchd ) );
209       iCheckEnergy++;
210     }
211     // To check stuff, has to be activated in the config file...
212        if (phases.p_en>=0) {
213         if (node==phases.p_en) {
214         fstream fEnergies;
215         char cFileName[50];
216         sprintf(cFileName, "Node%dEnergies.txt", phases.p_en);
217         fEnergies.open ( cFileName, fstream::out | fstream::app);
218         fEnergies << scientific << "Energy: " << e[0] << " " << e[1] << " " << e[2] << " " << e[3] << " " << e[0]-e[1] << " " <<
219 e[2]-e[3] << endl;
220         fEnergies << "Area: " << a[0] << " " << a[1] << " " << a[2] << " " << a[3] << " " << a[0]-a[1] << " " << a[2]-a[3] <<
221 endl;
222         fEnergies << "ClusterArea: " << ca[0] << " " << ca[1] << " " << ca[2] << " " << ca[3] << " " << ca[0]-ca[1] << " " <<
223 ca[2]-ca[3] << endl;
224         fEnergies.close();
225         //printf("Area:\t\t%le\t%le\t%le\t%le\n", a[0], a[1], a[2], a[3]);
226         //printf("ClusterArea:\t%le\t%le\t%le\t%le\n", ca[0], ca[1], ca[2], ca[3]);
227         }
228        }
229        return GetMoveDir( node, e[0]+a[0]+ca[0], e[1]+a[1]+ca[1], e[2]+a[2]+ca[2], e[3]+a[3]+ca[3], xy ,switchd);
230        //return GetMoveDir( node, e[0], e[1], e[2], e[3], xy ,switchd);
231 }
232
233
234 /* !/brief Runs through all the nodes (in random order) and moves them also writes out some statistics, turn
235 this on or off using STATS*/
236 int GBMGrowth()
237 {
238   int i, j, n, iMaxFlynns, iMaxNodes, movement, x;
239   int same=0, node_type;
240   Coords newxy;
241   vector < int > ran;
242
243   if ( (int) userdata[0] > 1 )
244     Settings_run.Count = (int) userdata[0];
245
246
247   if ( ElleCount() == 0 )
248     ElleAddDoubles();
249   if ( ElleDisplay() )
250     EllePlotRegions( ElleCount() );
251
252   ElleCheckFiles();
253
254   // Initialize the clusterTracking class...
255   clusterTracking clusters;
256   if ( clusters.writeInitialData("initial_stuff.txt") ) {
257     clusters.setClusterAreas();
258     clusters.checkDoubleClusterAreaLoop();
259     if ( phases.p_track == -1 )
260       UnodePhaseUpdate();
261   }
262
263
264
265   for ( i = 0; i < EllemaxStages(); i++ )
266   {
267     if ( bCheckEnergy == 1 ) {
268       if ( i == 1 )
269         bCheckEnergy = 2;
270     }
271     if ( !(i % 10) )
272       clusters.writeData( "PhaseAreaHistory.txt", i ); //cout << "STEP: " <<  i  << "/" << EllemaxStages() << endl;
273     iMaxNodes = ElleMaxNodes();
274
275     //to prevent moving a single node always at the same time, we shuffel them randomly at each step
276     ran.clear();
277     for ( j = 0; j < iMaxNodes; j++ )
278       if ( ElleNodeIsActive( j ) )
279         ran.push_back( j );
280     std::random_shuffle( ran.begin(), ran.end() );
281     for ( j = 0; j < ran.size(); j++ )
282     {
283       //cout << ran.at( j ) << endl;
284       //cout << "Start" << endl;
285       if ( ElleNodeIsActive( ran.at( j ) ) )
286       {
287         if (GGMoveNode( ran.at( j ), &newxy, &clusters ) ) { // same==1 &&
288           if (sqrt((newxy.x*newxy.x)+(newxy.y*newxy.y)) > 0.01)
289             cout << "PROBLEM: Movement is very large... Reduce mobility, energy or time step settings..." << endl;
290
291           ElleSetNodeChange(0);
292           ElleCrossingsCheck( ran.at( j ), & newxy );
293           if (ElleNodeChange() != 0)
294             clusters.updateClusters();
295
296           if (ElleNodeIsActive(j)) {
297             if ( ElleNodeIsDouble( j ) )
298               node_type = 2;
```

```
299             else if ( ElleNodeIsTriple( j ) )
300               node_type = 3;
301             else {
302               node_type = 0;
303               cout << "ERROR: No known node type(2)... (Node: " << j << ")" << endl;
304             }
305             if (node_type == 2 || node_type == 3) {
306               if (node_type == 2) {
307                 ElleSetNodeChange(0);
308                 ElleCheckDoubleJ( j );
309                 // Probably not neccessary since the only thing CheckDJ does is adding or removing dJs next to the node...
310                 if (ElleNodeChange() != 0)
311                   clusters.updateClusters();
312               }
313               else {
314                 // Checks if sufficient Triple nodes are available. Variable has to be set in the beginning of this file.
315                 // If the flynn has less TJs than this number no Triple J check will be done.
316                 // Possibly prevents statistical errors because of increasing flynn numbers during static grain growth.
317                 ///JR CHANGE TO THE TYPICAL ONCE IMPLEMENTED IN THE BASECODE AND ONCE IT IS POSSIBLE TO CHECK TRIPLE J IF
318     SOMETHING HAS CHANGED.
319                 if ( iMinTjs > 2 ) {
320                   int iNeighbours[3], iFlynns[3];
321                   int iNodeCount = 0;
322                   int * iNodes = NULL;
323                   int iTripleCheck;
324
325                   ElleNeighbourNodes( j, iNeighbours );
326                   for ( int k = 0; k < node_type; k++ ) {
327                       ElleNeighbourRegion( j, iNeighbours[k], &iFlynns[k] );
328                       ElleFlynnNodes( iFlynns[k], &iNodes, &iNodeCount );
329                       iTripleCheck = 0;
330                       for ( int l = 0; l < iNodeCount; l++ ) {
331                                 if ( ElleNodeIsTriple ( iNodes[l] ) ) {
332                                           iTripleCheck++;
333                                 }
334                       }
335
336                       if ( iTripleCheck < iMinTjs ) {
337                                 fstream fTripleFile;
338                                 fTripleFile.open ( "FailedTripleSwitches.txt", fstream::out | fstream::app);
339                                 if (fTripleFile.is_open()) {
340                                           fTripleFile << i << ": " << j << "{" << iTripleCheck << "} (" << iNeighbours[k] << ")
341     [" << iFlynns[k] << "]" << endl;
342                                           fTripleFile.close();
343                                 }
344                                 break;
345                       }
346                       free( iNodes );
347                       iNodes = NULL;
348                   }
349                   if ( iTripleCheck >= iMinTjs ) {
350                       ElleSetNodeChange(0);
351                       ElleCheckTripleJ( j );
352                       if (ElleNodeChange() != 0)
353                                 clusters.updateClusters();
354                   }
355                 }
356                 else {
357                   ElleSetNodeChange(0);
358                   ElleCheckTripleJ( j );
359                   if (ElleNodeChange() != 0)
360                       clusters.updateClusters();
361                 }
362               }
363             }
364           }
365         }
366       }
367     }
368     ///JR THIS ADDIDITIONAL CHECKING LOOP IS COMMENTED OUT FOR SPEED ATM. DON'T KNOW IF IT IS NECCESSARY ANYWAY...
369     //cout << "Loop End, Cluster...";
370     //clusters.updateClusters();
371     //cout << " ...check" << endl;
372     ////cout << "Loop finished" << endl;
373     //iMaxNodes = ElleMaxNodes();
374     //cout << "Checkloop...";
375     //for (j=0;j<iMaxNodes;j++) {
376       //if (ElleNodeIsActive(j)) {
377         //if ( ElleNodeIsDouble( j ) )
378           //node_type = 2;
379         //else if ( ElleNodeIsTriple( j ) )
380           //node_type = 3;
381         //else {
382           //node_type = 0;
383           //cout << "ERROR: No known node type(2)... (Node: " << j << ")" << endl;
384         //}
385         //if (node_type == 2 || node_type == 3) {
386           //if (node_type == 2) {
387             //ElleCheckDoubleJ( j );
388             ////clusters.updateClusters();
389           //}
390           //else {
391             //if ( iMinTjs > 2 ) {
392               //int iNeighbours[3], iFlynns[3];
393               //int iNodeCount = 0;
394               //int * iNodes = NULL;
395               //int iTripleCheck;
396
397               //ElleNeighbourNodes( j, iNeighbours );
398               //for ( int k = 0; k < node_type; k++ ) {
399                 //ElleNeighbourRegion( j, iNeighbours[k], &iFlynns[k] );
```

```
400               //ElleFlynnNodes( iFlynns[k], &iNodes, &iNodeCount );
401               //iTripleCheck = 0;
402               //for ( int l = 0; l < iNodeCount; l++ ) {
403                 //if ( ElleNodeIsTriple ( iNodes[l] ) ) {
404                   //iTripleCheck++;
405                 //}
406               //}
407
408               //if ( iTripleCheck < iMinTjs ) {
409                 //fstream fTripleFile;
410                 //fTripleFile.open ( "FailedTripleSwitches.txt", fstream::out | fstream::app);
411                 //if (fTripleFile.is_open()) {
412                   //fTripleFile << i << ": " << j << "{" << iTripleCheck << "} (" << iNeighbours[k] << ")  [" << iFlynns[k] <<
413     "]" << endl;
414                   //fTripleFile.close();
415                 //}
416                 //break;
417               //}
418               //free( iNodes );
419               //iNodes = NULL;
420             //}
421             //if ( iTripleCheck >= iMinTjs ) {
422               //ElleCheckTripleJ( j );
423               //clusters.updateClusters();
424             //}
425           //}
426           //else {
427             //ElleCheckTripleJ( j );
428             //clusters.updateClusters();
429           //}
430         //}
431       //}
432     //}
433   //}
434
435     // Just to be sure...
436     ElleAddDoubles();
437
438     // This does the cluster tracking using unodes...
439     if ( phases.p_track >= 0 ) // größer als max phases ist schon beim Auslesen berücksichtigt.
440       AssignUnodeProperties(); // Call the melt tracking.
441     else if ( phases.p_track == -1 ) {
442       // UnodePhaseShift();
443       UnodePhaseUpdate();
444       int iMaxUnodes = ElleMaxUnodes();
445       fstream fUnodePhases;
446       int iUPhases[3];
447       double iUConc[3];
448       fUnodePhases.open ( "UnodePhase.txt", fstream::out | fstream::app);
449
450       for ( int i = 0; i < 3; i++ ) {
451         iUPhases[i] = 0;
452         iUConc[i] = 0.0;
453       }
454
455       for ( int i = 0; i < iMaxUnodes; i++ ) {
456         double dPhase, dConc;
457         ElleGetUnodeAttribute( i, iUnodePhase, &dPhase );
458         ElleGetUnodeAttribute( i, iUnodeConc, &dConc );
459
460         if ( (int) dPhase == 0 || (int) dPhase == 1  ) {
461           iUPhases[ (int) dPhase ] += 1;
462           iUConc[ (int) dPhase ] += dConc;
463         }
464         else {
465           iUPhases[ 2 ] += 1;
466           iUConc[ 2 ] += dConc;
467         }
468       }
469       fUnodePhases << iMaxUnodes << "\t: " << iUConc[0]+iUConc[1]+iUConc[2] << "\t| " << iUPhases[0] << "\t: " << iUConc[0] <<
470     "\t| " << iUPhases[1] << "\t: " << iUConc[1] << "\t| " << iUPhases[2] << "\t: " << iUConc[2] << "\t| " << endl;
471       fUnodePhases.close();
472     }
473
474     ElleUpdate();
475   }
476 }
477
478 int UnodePhaseShift()
479 {
480   int iMaxUnodes = ElleMaxUnodes();
481   int iUnodesRow = (int) sqrt( (double) iMaxUnodes );
482   double dUnodeSpacing = 1.0 / (double) iUnodesRow;
483   Coords ref, xy;
484
485   for ( int i = 0; i < iMaxUnodes; i++ ) {
486     int iFlynn = ElleUnodeFlynn( i );
487     if ( ElleFlynnIsActive( iFlynn ) ) {
488       double dPhase, dOldPhase;
489       ElleGetFlynnRealAttribute( iFlynn, &dPhase, iFlynnPhase);
490       ElleGetUnodeAttribute( i, iUnodePhase, &dOldPhase );
491       if ( (int) dPhase != (int) dOldPhase ) {
492         cout << i << "(" << dOldPhase << ") "<< iFlynn << "(" << dPhase << ") " << endl;
493         int iFound = 0;
494         // first neighbours check
495         for ( int k = 0; k < 4 && iFound == 0; k ++ ) {
496           //0 --> left, 1 --> right, 2 --> down, 3 --> up
497           int iUnodeCheck;
498           if ( k == 0 ) {
499             // to the left of the unode, be careful of the left boundary
500             if ( !( i % iUnodesRow ) )
```

```
501               iUnodeCheck = i + iUnodesRow - 1;
502             else
503               iUnodeCheck = i - 1;
504           }
505           else if ( k == 1 ) {
506             // to the right of the unode, be careful of the right boundary
507             if ( !( ( i + 1 ) % iUnodesRow ) )
508               iUnodeCheck = i - iUnodesRow + 1;
509             else
510               iUnodeCheck = i + 1;
511           }
512           else if ( k == 2 ) {
513             // lower boundary, careful of the first row...
514             if ( i < iUnodesRow )
515               iUnodeCheck = i - iUnodesRow + iMaxUnodes;
516             else
517               iUnodeCheck = i - iUnodesRow;
518           }
519           else {
520             // upper boundary, careful of the last row...
521             if ( i >= iMaxUnodes - iUnodesRow )
522               iUnodeCheck = i + iUnodesRow - iMaxUnodes;
523             else
524               iUnodeCheck = i + iUnodesRow;
525           }
526
527           // Check the distance
528           ElleGetUnodePosition( i, &ref );
529           ElleGetUnodePosition( iUnodeCheck, &xy );
530           ElleCoordsPlotXY ( &ref, &xy );
531           double dist = pointSeparation( &ref, &xy );
532
533           if ( dist > dUnodeSpacing * 1.2 )
534             cout << "WARNING: UnodePhaseShift: Distance between Unodes too large... " << i << ":" << iUnodeCheck << " | " <<
535     dist << ">" << dUnodeSpacing * 1.2 << endl;
536
537
538           double dOldNeighbourPhase;
539           ElleGetUnodeAttribute( iUnodeCheck, iUnodePhase, &dOldNeighbourPhase );
540
541           //check if both were the same phase last step
542           if ( dOldPhase == dOldNeighbourPhase ) {
543             // if they were both the same shift the phase contents from one to the other
544             double dPhaseConc1, dPhaseConc2;
545             ElleGetUnodeAttribute( i, iUnodeConc, &dPhaseConc1 );
546             ElleGetUnodeAttribute( iUnodeCheck, iUnodeConc, &dPhaseConc2 );
547             ElleSetUnodeAttribute( i, iUnodeConc, 0.0 );
548             ElleSetUnodeAttribute( iUnodeCheck, iUnodeConc, ( dPhaseConc1 + dPhaseConc2 ) );
549             // The unode which changed phase gets set to 0.0
550             // the content gets transfered to the neighbour one.
551
552             iFound = 1; // break condition for the loop.
553           }
554
555           if ( k == 3 && iFound == 0 )
556             cout << "ERROR: UnodePhaseShift: No neighbour found!!!" << endl;
557         }
558       }
559     }
560     else
561       cout << "ERROR: UnodePhaseShift: Unode in inactive Flynn" << endl;
562   }
563 }
564
565 int UnodePhaseUpdate()
566 {
567   int iMaxFlynns = ElleMaxFlynns();
568   vector<int> vUnodes;
569
570   for ( int i = 0; i < iMaxFlynns; i++ ) {
571     if ( ElleFlynnIsActive( i ) ) {
572       ElleGetFlynnUnodeList( i, vUnodes );
573       double dPhase;
574       ElleGetFlynnRealAttribute( i, &dPhase, iFlynnPhase);
575       while ( vUnodes.size() > 0 ) {
576         ElleSetUnodeAttribute( vUnodes.back(), iUnodePhase, dPhase );
577         vUnodes.pop_back();
578       }
579     }
580   }
581   return 1;
582 }
583
584 // Written by Enrique & me to track changes in melt.
585 int AssignUnodeProperties()
586 {
587 int i, j, k, t; // Variables for looping
588    int max_flynns, max_unodes, flynnid, count;// num_nbs; // Variables to manipulate flynns, Unodes and their neighbours
589
590    ElleCheckFiles();
591
592    max_flynns = ElleMaxFlynns();
593    max_unodes = ElleMaxUnodes();
594
595   // Define array for unode property
596   double UnodeProp[max_unodes][3];
597
598   // Define array for flynn phase
599   double FlynnPhase[max_flynns];
600
601    //
```

```
602       // Loop to find Unodes and to store Unode and segment properties into the defined arrays
603       //
604       for (j=0;j<max_flynns;j++)
605       {
606            if (ElleFlynnIsActive(j))
607          {
608     //                ElleClearTriAttributes();
609     //                TriangulateUnodes(j,MeshData.tri); // Do the triangulation of the active flynn
610
611        vector<int> unodelist;  // create a vector list of unodes
612
613        ElleGetFlynnUnodeList(j,unodelist); // get the list of unodes for a flynn
614
615        ElleGetFlynnRealAttribute(j, &FlynnPhase[j], iFlynnPhase); // Read the phase of each flynn
616
617        count = unodelist.size(); // Number of unodes in flynn
618
619         for (i=0; i<count; i++)
620         {
621     //      vector<int> nbnodes,bndflag; // Define vectors
622
623     //      ElleGetTriPtNeighbours(unodelist[i],nbnodes,bndflag,0); // Get the  list of Unode neighbours of the selected Unode
624     and save it into nbnodes array
625     //      num_nbs = nbnodes.size(); // Number of neighbours of selected unode
626
627           ElleGetUnodeAttribute(unodelist[i],&UnodeProp[unodelist[i]][0],U_ATTRIB_A); // Read value of Unode property
628           ElleGetUnodeAttribute(unodelist[i],&UnodeProp[unodelist[i]][1],U_ATTRIB_B);
629           ElleGetUnodeAttribute(unodelist[i],&UnodeProp[unodelist[i]][2],U_ATTRIB_C);
630
631           if ( FlynnPhase[j] == phases.p_track ) // if Flynn is solid
632           {
633             UnodeProp[unodelist[i]][0] += 1; // New value of Unode property
634
635             if ( UnodeProp[unodelist[i]][2] != phases.p_track ) {
636               UnodeProp[unodelist[i]][1] += 1;
637             }
638           }
639
640           ElleSetUnodeAttribute(unodelist[i],UnodeProp[unodelist[i]][0],U_ATTRIB_A); // We assign a property to Unodes, depending
641     on to which flynn they belong
642           ElleSetUnodeAttribute(unodelist[i],UnodeProp[unodelist[i]][1],U_ATTRIB_B);
643           ElleSetUnodeAttribute(unodelist[i],FlynnPhase[j],U_ATTRIB_C);
644
645         }
646            }
647        }
648     }
649
650     int Read2PhaseDb(char *dbfile, AllPhases *phases)
651     {
652       fstream file;
653       string line;
654       stringstream linestr;
655       int no_phases=0, comb, count, p1, p2, infinite, cluster, diff_times, x, merge, p_en, el, scale, p_track;
656       double mob, en, dif, kappa, dGbActEn;
657       int input=0;
658       int plot=1; // enable plotting to command line.
659       char c;
660       // This functions reads the config file to storage...
661
662       file.open ("phase_db.txt", fstream::in );
663
664       if (file.is_open()) {
665         while (file.good()) {
666           getline (file,line);
667           if (line.length() > 0) {
668             // check for keywords
669             if ( line.find("PHASE PROPERTIES") != string::npos )
670               input = 1;
671             else if ( line.find("PHASE BOUNDARY PROPERTIES") != string::npos )
672               input = 2;
673             else if ( line.find("MELT TRACKING") != string::npos )
674               input = 3;
675             else if ( line.find("VERBOSE STUFF") != string::npos )
676               input = 4;
677             else if ( line.find("CLUSTER_TRACKING") != string::npos )
678               input = 5;
679             else if ( line.find("TROUBLESHOOTING") != string::npos )
680               input = 6;
681             c = line.at(0);
682             if (c!='#' && c!=' ') {
683               // Read number of phases
684               if ( input == 0 ) {
685                 linestr << line;
686                 linestr >> no_phases;
687                 linestr.clear();
688                 if (no_phases > MAX_PHASES) {
689                   cerr << "More phases than this program can handle" << endl;
690                   return 0;
691                 }
692                 phases->no_phases=no_phases;
693               }
694               // Read Phase properties
695               else if ( input == 1 ) {
696                 linestr << line;
697                 linestr >> p1 >> infinite >> cluster >> diff_times >> el >> scale >> kappa >> merge;
698                 linestr.clear();
699
700                 phases->phasep[p1].infinite_diff=infinite;
701                 phases->phasep[p1].cluster_diff=cluster;
702                 phases->phasep[p1].diffusion_times=diff_times;
```

```
703             phases->phasep[p1].elasticity=el;
704             phases->phasep[p1].scale=scale;
705             phases->phasep[p1].kappa=kappa;
706             phases->phasep[p1].merge=merge;
707           }
708           // Read Phase Boundary properties
709           else if ( input == 2 ) {
710             linestr << line;
711             linestr >> p1 >> p2 >> mob >> en >> dGbActEn;
712             linestr.clear();
713
714             phases->pairs[p1][p2].mobility=mob;
715             phases->pairs[p2][p1].mobility=mob;
716             phases->pairs[p1][p2].b_energy=en;
717             phases->pairs[p2][p1].b_energy=en;
718             phases->pairs[p1][p2].dGbActEn=dGbActEn;
719             phases->pairs[p2][p1].dGbActEn=dGbActEn;
720           }
721           // Read Phase Tracking
722           else if ( input == 3 ) {
723             linestr << line;
724             linestr >> p_track;
725             linestr.clear();
726             if ( p_track > no_phases-1 ) // wenn größer als max phases -> kein tracking (kleiner spielt keine Rolle)
727               p_track = -1;
728             phases->p_track=p_track;
729           }
730           // Read Verbouse
731           else if ( input == 4 ) {
732             linestr << line;
733             linestr >> p_en;
734             linestr.clear();
735
736             phases->p_en=p_en;
737           }
738           // read Clustertracking
739           // Read Troubleshooting
740           else if ( input == 6 ) {
741             linestr << line;
742             linestr >> iMinTjs;
743             linestr.clear();
744           }
745         }
746       }
747     }
748     // Plot the whole stuff into command line window.
749     if (plot == 1) {
750       cout << endl << "=================== INPUT FILE ===================" << endl
751         << "Number of phases: " << phases->no_phases << endl
752         << "==================== TIMESTEP ====================" << endl
753         << ElleTimestep() << " sec  <=>  " << ElleTimestep()/(60*60) << " h  <=>  "<< ElleTimestep()/(365*24*60*60) << " years"
754 << endl
755         << "================== LENGTH SCALE ==================" << endl
756         << ElleUnitLength() << " m  <=>  " << ElleUnitLength()*1000 << " mm" << endl //"  <=>  "<<
757 ElleTimestep()/(365*24*60*60) << " years" << endl
758         << "===================== PHASES =====================" << endl
759         << "Phase\tI-diff\tC-diff\tT-diff\telasticity\tscale\tkappa\tmerge" << endl;
760       for ( int i = 0; i < no_phases; i++ ) {
761         cout << i << "\t"
762           << phases->phasep[i].infinite_diff << "\t"
763           << phases->phasep[i].cluster_diff << "\t"
764           << phases->phasep[i].diffusion_times << "\t"
765           << phases->phasep[i].elasticity << "\t\t"
766           << phases->phasep[i].scale << "\t"
767           << phases->phasep[i].kappa << "\t"
768           << phases->phasep[i].merge
769           << endl;
770       }
771       cout << "================ PHASE BOUNDARIES ================" << endl
772         << "Phase1\tPhase2\tMobility\tB-energy\tGB-ActivEn" << endl;
773       for ( int i = 0; i < no_phases; i++ ) {
774         for ( int j = i; j < no_phases; j++ ) {
775           cout << i << "\t"
776             << j << "\t"
777             << phases->pairs[i][j].mobility << "\t\t"
778             << phases->pairs[i][j].b_energy << "\t\t"
779             << phases->pairs[i][j].dGbActEn
780             << endl;
781         }
782       }
783       if ( phases->p_track >= 0 ) {
784         cout
785           << "================== PHASE TRACKING ==================" << endl
786           << "Track phase: " << phases->p_track << " in Unode layer"
787           << endl;
788       }
789       if ( phases->p_en > 0 ) {
790         cout
791           << "================ VERBOSE STUFF ================" << endl
792           << "Print energies for node: " << phases->p_en
793           << endl;
794       }
795       if ( iMinTjs > 2 ) {
796         cout
797           << "================ TROUBLESHOOTING ================" << endl
798           << "Min Tjs: " << iMinTjs << " for Triple switching"
799           << endl;
800       }
801     }
802     file.close();
803     return (1);
```

```
804       } else
805         return (0);
806
807     }
808
809     // This returns the value for mob/en/any other stuff defined by type for a segment between node 1 and node 2
810     // type = 0 == mobility, 1==b_energy, 2==Activation Energy
811     double CheckPair(int node1, int node2, int type)
812     {
813       int rgn[2], int_a[2], i;
814       double type_a[2];
815
816       ElleNeighbourRegion(node1,node2,&rgn[0]);
817       ElleNeighbourRegion(node2,node1,&rgn[1]);
818       ElleGetFlynnRealAttribute(rgn[0], &type_a[0], iFlynnPhase);
819       ElleGetFlynnRealAttribute(rgn[1], &type_a[1], iFlynnPhase);
820       //printf("%lf %lf\n", type_a[0], type_a[1]);
821       int_a[0] = (int)type_a[0];
822       int_a[1] = (int)type_a[1];
823
824       //  printf("%le %le %d %d\n", phases.pairs[int_a[0]][int_a[1]].mobility,
825     phases.pairs[int_a[0]][int_a[1]].b_energy,int_a[0],int_a[1]);
826       if (type == 0)
827         return phases.pairs[int_a[0]][int_a[1]].mobility;
828       else if (type == 1)
829         return phases.pairs[int_a[0]][int_a[1]].b_energy;
830       else if (type == 2)
831         return phases.pairs[int_a[0]][int_a[1]].dGbActEn;
832       else
833         return 0;
834
835     }
836
837     int StoreAreaChange(int node, Coords *vector, int node_type)
838     {
839       int  type_i[3], x, nghbr[3], n, j, nnode[3], rgn[3];
840       double area[9], phase_area[phases.no_phases], bla, type_a[3];
841       Coords loc;
842
843       // find the neighbours
844       if (ElleNeighbourNodes(node,nghbr))
845         cout << "StoreAreaChange: Neighbour determination..." << endl;
846       // read the attributes of that node
847       for (n=0,j=0;n<3;n++)
848       {
849         if(nghbr[n]!=NO_NB && j<node_type) {
850           nnode[j] = nghbr[n];
851           ElleNeighbourRegion(node,nghbr[n],&rgn[j]);
852           ElleGetFlynnRealAttribute(rgn[j], &type_a[j], iFlynnPhase);
853           type_i[j] = (int)type_a[j];
854           j++;
855         }
856       }
857
858       ElleNodePosition(node, &loc);
859
860       loc.x = loc.x+vector->x;
861       loc.y = loc.y+vector->y;
862
863       if (node_type == 2) {
864         //do nothing if it is a same phase everywhere node
865         if (type_i[0]==type_i[1])
866           ;
867         //if there are different phases store the areas.
868         else {
869           x=GetArea(node, area, phase_area, type_i, node_type, &loc);
870     //      cout << "node: " << node << endl;
871     //      cout << "area:";
872     //      for (int l = 0; l < 9; l ++ )
873     //       cout << " " << area[l];
874     //      cout << endl;
875     //      cout << "phase area-area:";
876     //      for (int l = 0; l < phases.no_phases; l++)
877     //        cout << " " << phase_area[l] << "-" << type_i[l];
878     //      cout << endl << "node type: " << node_type << endl;
879
880           if(x==0)
881             printf("ERROR: StoreAreaChange: node_type=2: GetArea");
882
883           ElleSetNodeAttribute(node, (phase_area[type_i[0]]+area[2]), attrib[type_i[0]]);
884           //this check is needed, because if both are the same phase the addition wouldn't be considered and just overwritten
885           if (type_i[0] == type_i[1])
886             phase_area[type_i[1]] = ElleNodeAttribute(node, attrib[type_i[0]]);
887           ElleSetNodeAttribute(node, (phase_area[type_i[1]]-area[2]), attrib[type_i[1]]);
888         }
889       }
890       else if (node_type == 3) {
891         //do nothing if it is a same phase node
892         if (type_i[0]==type_i[1] && type_i[0]==type_i[2])
893           ;
894         else {
895           x=GetArea(node, area, phase_area, type_i, node_type, &loc);
896           if(x==0)
897             printf("ERROR: StoreAreaChange: node_type=3: GetArea");
898
899           ElleSetNodeAttribute(node, ((phase_area[type_i[0]])+area[6]), attrib[type_i[0]]);
900           // same as double node check...
901           if (type_i[0] == type_i[1])
902             phase_area[type_i[1]] = ElleNodeAttribute(node, attrib[type_i[1]]);
903           ElleSetNodeAttribute(node, ((phase_area[type_i[1]])+area[7]), attrib[type_i[1]]);
904           if (type_i[1] == type_i[2])
```

```
905             phase_area[type_i[2]] = ElleNodeAttribute(node, attrib[type_i[2]]);
906           if (type_i[0] == type_i[2])
907             phase_area[type_i[2]] = ElleNodeAttribute(node, attrib[type_i[2]]);
908           ElleSetNodeAttribute(node, ((phase_area[type_i[2]])+area[8]), attrib[type_i[2]]);
909         }
910       }
911       else
912         printf("Are you kidding??? What kind of node is it then?\n");
913
914       return 1;
915     }
916
917     int GetArea(int node, double area[], double phase_area[], int type_i[], int nodetype, Coords *loc)
918     {
919       int nghbr[3], i, j, rgn[3], err=0, x=1;
920       double type_a[3];
921       Coords location[5];
922
923       // store the location of the center node to location 3, and the new position at location 4
924       ElleNodePosition(node, &location[3]);
925       location[4].x = loc->x;
926       location[4].y = loc->y;
927
928       // store the locations of the imidiate neigbour nodes as location 0,1 and 2 if it is a Triple Node.
929       if (err=ElleNeighbourNodes(node,nghbr))
930         printf("ERROR: Get Area: getting neighbour nodes\n");
931       for (i=0, j=0;i<3;i++)
932       {
933           if(nghbr[i]!=NO_NB && j<nodetype) {
934     //      ElleNodePlotXY(nghbr[i], &location[j], &location[3]);
935           ElleNeighbourRegion(node,nghbr[i],&rgn[j]);
936           ElleGetFlynnRealAttribute(rgn[j], &type_a[j], iFlynnPhase);
937           type_i[j] = (int)type_a[j];
938           j++;
939         }
940       }
941       for (i=0;i<phases.no_phases;i++)
942         phase_area[i] = ElleNodeAttribute(node, attrib[i]);
943
944       if (nodetype == 2) {
945         if ( type_i[0] != type_i[1] ) {
946           area[0] = ElleRegionArea(rgn[0]);
947           area[1] = ReturnArea(rgn[0],node,&location[4]);
948           area[2] = area[1] - area[0];
949         } else
950           area[2] = 0;
951       }
952       else if (nodetype == 3) {
953         // if all 3 phases along the boardering the triple node are the same. -> 0 area change...
954         if ( type_i[0] == type_i[1] && type_i[0] == type_i[2] && type_i[1] == type_i[2]) {
955           area[6] = 0;
956           area[7] = 0;
957           area[8] = 0;
958         } else {
959           area[0] = ElleRegionArea(rgn[0]);
960           area[1] = ElleRegionArea(rgn[1]);
961           area[2] = ElleRegionArea(rgn[2]);
962           area[3] = ReturnArea(rgn[0],node,&location[4]);
963           area[4] = ReturnArea(rgn[1],node,&location[4]);
964           area[5] = ReturnArea(rgn[2],node,&location[4]);
965           area[6] = area[3] - area[0];
966           area[7] = area[4] - area[1];
967           area[8] = area[5] - area[2];
968           //cout << area[6] << " " << area[7] << " " << area[8] << endl;
969         }
970       }
971       else
972         printf("ERROR: GetArea: Are you kidding??? What kind of node is it then?\n");
973
974       if (err==1)
975         x=0;
976
977       return x;
978     }
979
980     double ReturnAreaEnergy(int node, Coords *loca)
981     {
982       int type_i[3], node_type, x;
983       double a[3], ret_area, area[9], phase_area[phases.no_phases], temp_area;
984
985       //set area to 0
986       a[0]=a[1]=a[2]=0;
987
988
989       if (ElleNodeIsDouble(node))
990         node_type = 2;
991       else if (ElleNodeIsTriple(node))
992         node_type = 3;
993       else
994         return 0;
995
996
997       if (node_type == 2) {
998
999         x=GetArea(node, area, phase_area, type_i, node_type, loca);
1000        if(x==0)
1001          printf("ERROR: ReturnAreaEnergy: node_type=2: GetArea");
1002
1003
1004        //a1
1005        if(phases.phasep[type_i[0]].elasticity==0.0)  //not used if elasticity[a1]=0.0
```

```
1006          a[0]=0.0;
1007        else {
1008          a[0]=(pow((fabs(phase_area[type_i[0]]+area[2])),(phases.phasep[type_i[0]].elasticity)))*phases.phasep[type_i[0]].scale;
1009    //    if (node==615)
1010    //      printf("%le,
1011    %le+%le^%le*%le\n",a[0],phase_area[type_i[0]],area[2],phases.phasep[type_i[0]].elasticity,phases.phasep[type_i[0]].scale);
1012        }
1013        //a2
1014        if(phases.phasep[type_i[1]].elasticity==0.0)  //not used if elasticity[a2]=0.0
1015          a[1]=0.0;
1016        else {
1017          a[1]=(pow((fabs(phase_area[type_i[1]]-area[2])),(phases.phasep[type_i[1]].elasticity)))*phases.phasep[type_i[1]].scale;
1018        }
1019
1020        //return both added together
1021        ret_area=a[0]+a[1];
1022      }
1023      else if (node_type == 3) {
1024
1025        x=GetArea(node, area, phase_area, type_i, node_type, loca);
1026        if(x==0)
1027          printf("ERROR: ReturnAreaEnergy: node_type=3: GetArea");
1028
1029        //a1
1030        if(phases.phasep[type_i[0]].elasticity==0.0)  //not used if elasticity[a1]=0.0
1031          a[0]=0.0;
1032        else {
1033          a[0]=(pow((fabs(phase_area[type_i[0]]+area[6])),(phases.phasep[type_i[0]].elasticity)))*phases.phasep[type_i[0]].scale;
1034        }
1035        //a2
1036        if(phases.phasep[type_i[1]].elasticity==0.0)  //not used if elasticity[a2]=0.0
1037          a[1]=0.0;
1038        else {
1039          a[1]=(pow((fabs(phase_area[type_i[1]]+area[7])),(phases.phasep[type_i[1]].elasticity)))*phases.phasep[type_i[1]].scale;
1040        }
1041        //a3
1042        if(phases.phasep[type_i[2]].elasticity==0.0)  //not used if elasticity[a3]=0.0
1043          a[2]=0.0;
1044        else {
1045          a[2]=(pow((fabs(phase_area[type_i[2]]+area[8])),(phases.phasep[type_i[2]].elasticity)))*phases.phasep[type_i[2]].scale;
1046        }
1047
1048        //return all three added together
1049        ret_area=a[0]+a[1]+a[2];
1050    //   cout << a[0] << "+" << a[1] << "+" << a[2] << "=" << ret_area << endl;
1051      }
1052      else
1053        printf("Are you kidding??? What kind of node is it then?\n");
1054
1055      return ret_area;
1056    }
1057
1058    double ReturnArea(ERegion poly, int node, Coords *pos)
1059    {
1060         int j, *id=0, num_nodes;
1061         double area, *coordsx=0, *coordsy=0, *ptrx, *ptry;
1062         Coords xy,prev;
1063         list<int> nodes;
1064         //printf("x:%f\ty:%f\n", pos->x, pos->y);
1065
1066         ElleFlynnNodes(poly,&id,&num_nodes);
1067         if ((coordsx = (double *)malloc(num_nodes*sizeof(double)))== 0) OnError("ElleRegionArea",MALLOC_ERR);
1068         if ((coordsy = (double *)malloc(num_nodes*sizeof(double)))== 0) OnError("ElleRegionArea",MALLOC_ERR);
1069
1070         for (j=0;j<num_nodes;j++)
1071          nodes.push_back(id[j]);
1072
1073         if (num_nodes != nodes.size())
1074          printf("ERROR: ReturnArea: sizes don't match\n");
1075
1076         if (id) free(id);
1077
1078         j=0;
1079
1080         //just reordeer the list until the current node is at the front
1081         while (j==0) {
1082          if(nodes.front()==node)
1083            j=1;
1084          else {
1085            nodes.push_back(nodes.front());
1086            nodes.pop_front();
1087          }
1088         }
1089
1090         //do the same stuff as ElleRegionArea except for the first "node"
1091         prev.x = pos->x;
1092         prev.y = pos->y;
1093
1094         ptrx=coordsx;
1095         ptry=coordsy;
1096         while (nodes.size()>0) {
1097          if (nodes.front()==node){
1098            xy=*pos;
1099            ElleCoordsPlotXY(&xy, &prev);
1100          *ptrx = xy.x; ptrx++;
1101          *ptry = xy.y; ptry++;
1102          nodes.pop_front();
1103          } else {
1104          ElleNodePlotXY(nodes.front(),&xy,&prev);
1105          *ptrx = xy.x; ptrx++;
1106          *ptry = xy.y; ptry++;
```

```
1107          prev = xy;
1108          nodes.pop_front();
1109          }
1110         }
1111         area = polyArea(coordsx,coordsy,num_nodes);
1112         free(coordsx);
1113         free(coordsy);
1114         return(area);
1115     }
1116
1117     clusterTracking::clusterTracking( void )
1118     {
1119
1120       // find the phases --> temporary since it has to stay compatible to other functions...
1121       lInfDiffPhases.clear();
1122       lClustDiffPhases.clear();
1123       lFicksDiffPhases.clear();
1124       lAllPhases.clear();
1125
1126       for ( int i = 0; i < phases.no_phases; i++ ) {
1127         if ( phases.phasep[ i ].infinite_diff == 1 )
1128           if ( phases.phasep[ i ].cluster_diff == 1 ) {
1129             lClustDiffPhases.push_back( i );
1130             lAllPhases.push_back( i );
1131             vClusterPhases.push_back( i );
1132           }
1133           else {
1134             lInfDiffPhases.push_back( i );
1135             lAllPhases.push_back( i );
1136           }
1137         else {
1138           lFicksDiffPhases.push_back( i );
1139           lAllPhases.push_back( i );
1140         }
1141       }
1142
1143       // read the Cluster Tracking stuff from the config file
1144
1145       bool bFound = false;
1146       fstream file;
1147       string line;
1148       stringstream linestr;
1149       char c;
1150       // This functions reads the config file to storage...
1151
1152       file.open ("phase_db.txt", fstream::in );
1153
1154       if (file.is_open()) {
1155         while (file.good()) {
1156           getline (file,line);
1157           //cout << line << endl;
1158           if ( bFound == false ) {
1159             if ( line.find("CLUSTER_TRACKING") != string::npos )
1160               bFound = true;
1161           }
1162           else {
1163             if (line.length() > 0) {
1164               c = line.at(0);
1165               if (c!='#' && c!=' ') {
1166                 linestr.clear();
1167                 linestr << line;
1168                 linestr >> dMultiplierA >> dMultiplierB >> dMultiplierC >> dMultiplierD;
1169                 linestr.clear();
1170                 bFound = false;
1171               }
1172             }
1173           }
1174         }
1175       }
1176       file.close();
1177
1178       // set the max Flynns parameter.
1179       dAreaShift = 1e-10;
1180       clusterTracking::findClusters();
1181     }
1182
1183     clusterTracking::~clusterTracking()
1184     {
1185     }
1186
1187     void clusterTracking::updateClusters( void )
1188     {
1189       clusterTracking::findClusters();
1190       clusterTracking::findSplit();
1191       clusterTracking::findMerge();
1192       clusterTracking::checkDoubleClusterAreaLoop();
1193     }
1194
1195     void clusterTracking::getPhaseAreas( void )
1196     {
1197       double dPhaseArea;
1198
1199       vPhaseAreas.clear();
1200       // get the complete area for each phase
1201       for ( int i = 0; i < phases.no_phases; i++ ) {
1202         dPhaseArea = 0; // set the start to 0
1203         for ( int j = 0; j < iMaxFlynns; j++ ) {
1204           if ( vFlynnPhase.at ( j ) == i ) {
1205             dPhaseArea += ElleRegionArea( vFlynns.at ( j ) );
1206           }
1207         }
```

```
1208        vPhaseAreas.push_back( dPhaseArea );
1209      }
1210    }
1211
1212    bool clusterTracking::writeInitialData( const char *filename )
1213    {
1214      fstream fInitial;
1215
1216      if (!fileExists( filename )) {
1217
1218        fInitial.open ( filename, fstream::out | fstream::trunc);
1219
1220        clusterTracking::getPhaseAreas();
1221
1222        if (fInitial.is_open()) {
1223          fInitial << scientific << vPhaseAreas[0];
1224          for ( int i = 1; i < phases.no_phases; i++ )
1225            fInitial << " " << vPhaseAreas[i];
1226          fInitial << endl;
1227        }
1228        fInitial.close();
1229
1230        return true;
1231      }
1232      else {
1233        cout << "WARNING: initial file present!" << endl << "If you start at step 0 delete this file first!" << endl;
1234        return false;
1235      }
1236    }
1237
1238    bool clusterTracking::writeData( const char *filename, int step)
1239    {
1240      fstream fDataFile;
1241
1242      fDataFile.open ( filename, fstream::out | fstream::app);
1243
1244      clusterTracking::getPhaseAreas();
1245
1246      if (fDataFile.is_open()) {
1247        fDataFile << step << " " << scientific << vPhaseAreas[0];
1248        for ( int i = 1; i < phases.no_phases; i++ )
1249          fDataFile << " " << vPhaseAreas[i];
1250        fDataFile << endl;
1251      }
1252      fDataFile.close();
1253
1254      return true;
1255    }
1256
1257
1258    void clusterTracking::getClusters( void )
1259    {
1260      int temp_int, iOnList;
1261      double temp_double;
1262
1263      vector<int> vCluster;
1264      vector<vector<int> > vClusters;
1265      list<int> lOriginal, lNeighbour;
1266
1267      for ( int z = 0; z < vClusterPhases.size(); z++ ) {
1268        lOriginal.clear();
1269        // get all flynns with phase i
1270        for ( int j = 0; j < iMaxFlynns; j++ ) {
1271          if ( vFlynnPhase.at ( j ) == vClusterPhases.at( z ) )
1272            lOriginal.push_back( vFlynns.at ( j ) );
1273        }
1274        // just to be sure that every Flynn is only once in the Vector.
1275        lOriginal.sort();
1276        lOriginal.unique();
1277
1278        // find flynns that are clustered together
1279        // as long as there are any flynns in the phase list
1280        while ( lOriginal.size() > 0 ) {
1281          vCluster.clear();
1282          vCluster.push_back( lOriginal.front() ); // put the first flynn into the cluster list
1283          lOriginal.pop_front(); // delete that element from the list
1284
1285          for ( int n = 0; n < vCluster.size(); n++ ) {
1286            lNeighbour.clear(); // clear the neigbour list
1287            ElleFlynnNbRegions( vCluster.at(n), lNeighbour ); //find neighbours for the current flynn (n) in the cluster list
1288
1289            // check whether any flynn in the neighbour list matches the current phase (i)
1290            // as long as there are entries in the neighbours list do the following
1291            while ( lNeighbour.size() > 0 ) {
1292              ElleGetFlynnRealAttribute( lNeighbour.front(), &temp_double, iFlynnPhase ); // get phase from flynn
1293              temp_int = (int) temp_double; // convert to int
1294
1295              //compare to current phase
1296              if ( temp_int == vClusterPhases.at( z ) ) {                 // if Flynn has the same phase
1297                // look whether the Flynn is already on the cluster list
1298                int iOnList = 0;
1299                for ( int i = 0; i < vCluster.size() && iOnList == 0; i++ )
1300                  if ( lNeighbour.front() == vCluster.at(i) )
1301                    iOnList = 1;
1302                if ( iOnList == 0 ) {                                                   // if NOT
1303                  vCluster.push_back( lNeighbour.front() ); // add flynn to cluster list
1304                  lOriginal.remove( lNeighbour.front() );         // remove that flynn from the original phase list
1305                  lNeighbour.pop_front();                                               // remove it from the neighbours list
1306                }
1307                else                                                                    // if it is
1308                  lNeighbour.pop_front();                                             // just remove it from the neighbours list
```

```
1309            }
1310            else                                                        // if Flynn has not the same phase
1311              lNeighbour.pop_front();                                   // just remove it from the neighbours list
1312          }
1313        }
1314        vClusters.push_back(vCluster);    //vector which contains all the flynns beloning to a cluster is put in another vector
1315      }
1316      vPhasesClusters.push_back( vClusters );
1317    }
1318
1319    //for ( int z = 0; z < vPhasesClusters.size(); z++ ) {
1320      //for ( int i = 0; i < vPhasesClusters[z].size(); i++ ) {
1321        //cout << vPhasesClusterAreas[z][i] << " :";
1322        //for ( int j = 0; j < vPhasesClusters[z][i].size(); j++ ) {
1323          //cout << " " << vPhasesClusters[z][i][j];
1324        //}
1325        //cout << endl;
1326      //}
1327    //}
1328  }
1329
1330  vector<double> clusterTracking::returnMultiplier ( vector<double> vAreaPercentage )
1331  {
1332    //cout << dMultiplierA << " " << dMultiplierB << " " << dMultiplierC << " " << dMultiplierD << endl;
1333    vector<double> vMultiplier;
1334
1335    for ( int i = 0; i < vAreaPercentage.size(); i++ ) {
1336      vMultiplier.push_back ( ( dMultiplierA * pow( vAreaPercentage[i], dMultiplierD ) ) + ( dMultiplierB * vAreaPercentage[i] )
1337  + dMultiplierC );
1338    }
1339    return vMultiplier;
1340  }
1341
1342  double clusterTracking::returnFlynnAreaChange ( int iFlynn, int iNode, Coords * xyNewPos )
1343  {
1344    //if (iNode == phases.p_en ) {
1345      //fstream file;
1346      //file.open ("2.txt", fstream::out | fstream::app );
1347      //file << ReturnArea( iFlynn, iNode, xyNewPos ) << " - " << ElleRegionArea( iFlynn ) << " = " << (ReturnArea( iFlynn,
1348  iNode, xyNewPos ) - ElleRegionArea( iFlynn ) ) << endl;
1349      //file.close();
1350    //}
1351    return ( ReturnArea( iFlynn, iNode, xyNewPos ) - ElleRegionArea( iFlynn ) );
1352  }
1353
1354  vector<double> clusterTracking::returnClusterAreaChange ( vector<vector<int> > vPhaseFlynns, int iNode, Coords * xyLoc )
1355  {
1356    bool bFound;
1357    double dClusterArea, dClusterAreaCheck;
1358    vector<double> vClusterAreaPercentageChange, vClusterAreaMultiplier, vClusterAreaEnergy, vCurrentArea, vClusterArea;// just
1359  get the area for all clusters which touch that node and store it there.
1360    vector<double> vPhaseClusterAreaChange; // get the area change for all clusters which touch the node with the given
1361  movement.
1362
1363    for ( int i = 0; i < vPhaseFlynns.size(); i++ ) {
1364      vPhaseClusterAreaChange.push_back ( 0 );
1365      ElleGetFlynnRealAttribute( vPhaseFlynns[i][0], &dClusterArea, iFlynnCluster );
1366      for ( int j = 0; j < vPhaseFlynns[i].size(); j++ ) {
1367        vPhaseClusterAreaChange.at ( i ) +=  returnFlynnAreaChange ( vPhaseFlynns[i][j], iNode, xyLoc );
1368        if ( j > 0 ) {
1369          ElleGetFlynnRealAttribute( vPhaseFlynns[i][j], &dClusterAreaCheck, iFlynnCluster );
1370          if ( dClusterArea != dClusterAreaCheck ) {
1371            cout << "WARNING: Stored Clusterareas in the Flynns that belong to the same Cluster are not the same!!" << endl;
1372            //cout << dClusterArea << "=" << dClusterAreaCheck << endl;
1373            //cout << *vPhaseFlynns[i][0] << " - " << *vPhaseFlynns[i][j] << endl;
1374            clusterTracking::findClusters();
1375            clusterTracking::findSplit();
1376            clusterTracking::findMerge();
1377            clusterTracking::checkDoubleClusterAreaLoop();
1378            ElleGetFlynnRealAttribute( vPhaseFlynns[i][0], &dClusterArea, iFlynnCluster );
1379          }
1380        }
1381      }
1382
1383      vClusterArea.push_back( dClusterArea );
1384
1385
1386      // get the current area
1387      for ( int z = 0, bFound = false; z < vPhasesClusters.size() && bFound == false; z++ ) {
1388        for ( int j = 0; j < vPhasesClusters[z].size() && bFound == false; j++ ) {
1389          for ( int k = 0; k < vPhasesClusters[z][j].size() && bFound == false; k++ ) {
1390            if ( vPhasesClusters[z][j][k] == vPhaseFlynns[i][0] ) {
1391              vCurrentArea.push_back( vPhasesClusterAreas[z][j] );
1392              bFound = true;
1393            }
1394          }
1395        }
1396      }
1397      //vClusterAreaPercentageChange.push_back ( fabs( ( ( vCurrentArea.at( i ) + vPhaseClusterAreaChange.at( i ) ) /
1398  vClusterArea.at( i ) ) - 1 ) );
1399    }
1400    //vClusterAreaMultiplier = clusterTracking::returnMultiplier ( vClusterAreaPercentageChange );
1401    if ( vPhaseClusterAreaChange.size() != vCurrentArea.size() )
1402      cout << "ERROR in returnClusterAReaChange --> Vector size (PhaseCluster:CurrentArea) Not the same!! "
1403        << vPhaseClusterAreaChange.size() << ":" << vCurrentArea.size()
1404        << " " << vPhaseClusterAreaChange.front() << " " << vPhaseFlynns[0][0] << endl;
1405
1406    if ( vPhaseClusterAreaChange.size() != vClusterArea.size() )
1407      cout << "ERROR in returnClusterAReaChange --> Vector size (PhaseCluster:ClusterArea) Not the same!!"
1408        << vPhaseClusterAreaChange.size() << ":" << vClusterArea.size() << endl;
1409
```

```
1410        for ( int i = 0; i < vPhaseClusterAreaChange.size(); i++ ) {
1411          vClusterAreaEnergy.push_back ( dMultiplierA * pow ( fabs( ( ( vCurrentArea.at( i ) + ( vPhaseClusterAreaChange.at( i ) ) )
1412      - vClusterArea.at( i ) ) / vClusterArea.at( i ) ), dMultiplierD ) );
1413          //vClusterAreaEnergy.push_back ( fabs( vCurrentArea.at( i ) + ( vPhaseClusterAreaChange.at( i ) ) - vClusterArea.at( i ) )
1414      * vClusterAreaMultiplier.at( i ) );
1415        }
1416
1417
1418        if (iNode == phases.p_en ) {
1419          fstream file;
1420          file.open ("1.txt", fstream::out | fstream::app );
1421          for ( int i = 0; i < vPhaseClusterAreaChange.size(); i++ ) {
1422            file << "TripleNodeArea " << i << " " << xyLoc->x << " " << xyLoc->y << " " << vPhaseClusterAreaChange.at ( i ) << " "
1423      << vClusterArea.at( i ) << " " << vCurrentArea.at( i ) << " " << vClusterAreaEnergy.at ( i ) << endl;
1424           //file << "TripleNodeArea " << i << " " << xyLoc->x << " " << xyLoc->y << " " << vPhaseClusterAreaChange.at ( i ) << " "
1425      << vClusterArea.at( i ) << " " << vCurrentArea.at( i ) << " " << vClusterAreaPercentageChange.at ( i ) << " " <<
1426      vClusterAreaMultiplier.at ( i ) << " " << vClusterAreaEnergy.at ( i ) << endl;
1427          }
1428          file.close();
1429        }
1430
1431        //if (iNode == 18474 ) {
1432          //for ( int i = 0; i < vClusterAreaPercentageChange.size(); i++ ) {
1433            //cout << "TripleNode " << i << " : " << vClusterAreaPercentageChange.at ( i ) << " : " << vClusterAreaMultiplier.at ( i
1434      ) << " : " << vClusterAreaEnergy.at ( i ) << endl;
1435          //}
1436        //}
1437
1438        //if (iNode == 14034 ) {
1439          //for ( int i = 0; i < vClusterAreaPercentageChange.size(); i++ ) {
1440            //cout << "DoubleNode " << i << " : " << vClusterAreaPercentageChange.at ( i ) << " : " << vClusterAreaMultiplier.at ( i
1441      ) << " : " << vClusterAreaEnergy.at ( i ) << endl;
1442          //}
1443        //}
1444
1445        return vClusterAreaEnergy;
1446      }
1447
1448      double clusterTracking::returnClusterAreaEnergy ( int iNode , Coords * xyLoc )
1449      {
1450        int iNodeType = 3;
1451        int iNeighbours[3], iFlynns[3];
1452
1453        ElleNeighbourNodes(iNode,iNeighbours);
1454
1455        //cout << "Node: " << iNode << ", Neighbours:";
1456        //for ( int i = 0; i < 3; i++ )
1457          //cout << " " << iNeighbours[i];
1458        //cout << endl;
1459
1460        for ( int i = 0, j = 0; i < 3; i++ ) {
1461          if ( iNeighbours[i] != NO_NB ) {
1462            ElleNeighbourRegion(iNode,iNeighbours[i],&iFlynns[j]);
1463            j++;
1464          }
1465          else
1466            iNodeType = 2;
1467        }
1468        //cout << "NodeType: " << iNodeType << ", NeighbourFlynns:";
1469        //for ( int i = 0; i < iNodeType; i++ )
1470          //cout << " " << iFlynns[i];
1471        //cout << endl;
1472
1473
1474
1475        int iFlynnPhaseCheck;
1476        double dFlynnPhaseCheck;
1477        vector<int> vClusterPhaseFlynns;
1478        vector<vector<int> > vPhaseClusterFlynns;
1479
1480        for ( int i = 0; i < vClusterPhases.size(); i++ ) {
1481          vClusterPhaseFlynns.clear();
1482          for ( int j = 0; j < iNodeType; j++ ) {
1483            ElleGetFlynnRealAttribute( iFlynns[j], &dFlynnPhaseCheck, iFlynnPhase );
1484            iFlynnPhaseCheck = (int) dFlynnPhaseCheck;
1485            if ( iFlynnPhaseCheck == vClusterPhases[ i ] ) {
1486              vClusterPhaseFlynns.push_back( iFlynns[j] );
1487            }
1488          }
1489          if ( vClusterPhaseFlynns.size() > 0 )
1490            vPhaseClusterFlynns.push_back( vClusterPhaseFlynns );
1491        }
1492
1493        if ( vPhaseClusterFlynns.size() == 0 )
1494          return 0.0;
1495
1496        double dClusterAreaEnergy = 0;
1497        vector<double> vClusterAreaEnergy = clusterTracking::returnClusterAreaChange ( vPhaseClusterFlynns, iNode, xyLoc );
1498
1499        for ( int i = 0; i < vClusterAreaEnergy.size(); i++ )
1500          dClusterAreaEnergy += vClusterAreaEnergy.at ( i );
1501
1502        return dClusterAreaEnergy;
1503      }
1504
1505
1506      void clusterTracking::getClusterAreas( void )
1507      {
1508        // Calculate the Area of the Cluster
1509        double dClusterArea = 0.0;
1510
```

```
1511      vector<double> vClusterArea;
1512
1513      vPhasesClusterAreas.clear();
1514
1515      for ( int z = 0; z < vPhasesClusters.size(); z++ ) {
1516        vClusterArea.clear();
1517        for ( int i = 0; i < vPhasesClusters[z].size(); i++ ) {
1518          dClusterArea = 0.0;
1519          for ( int j = 0; j < vPhasesClusters[z][i].size(); j++ ) {
1520            dClusterArea += ElleRegionArea( vPhasesClusters[z][i][j] );
1521          }
1522          vClusterArea.push_back( dClusterArea );
1523        }
1524        vPhasesClusterAreas.push_back( vClusterArea );
1525      }
1526    }
1527
1528    void clusterTracking::setClusterAreas( void )
1529    {
1530      for ( int z = 0; z < vPhasesClusters.size(); z++ ) {
1531        for ( int i = 0; i < vPhasesClusters[z].size(); i++ ) {
1532          for ( int j = 0; j < vPhasesClusters[z][i].size(); j++ ) {
1533            ElleSetFlynnRealAttribute( vPhasesClusters[z][i][j], vPhasesClusterAreas[z][i], iFlynnCluster );
1534          }
1535        }
1536      }
1537    }
1538
1539    void clusterTracking::findSplit( void )
1540    {
1541      double dArea, dAreaCheck;
1542
1543      vector<int> vSplitClusterFlynns;
1544      vector<vector<int> > vSplitClusters;
1545
1546      for ( int z = 0; z < vPhasesClusters.size(); z++ ) {
1547        for ( int i = 0; i < vPhasesClusters[z].size(); i++ ) {
1548          ElleGetFlynnRealAttribute( vPhasesClusters[z][i][0], &dArea, iFlynnCluster );
1549
1550          vSplitClusters.clear();
1551
1552          for ( int j = 0; j < vPhasesClusters[z].size(); j++ ) {
1553            vSplitClusterFlynns.clear();
1554            for ( int k = 0; k < vPhasesClusters[z][j].size(); k++ ) {
1555              ElleGetFlynnRealAttribute( vPhasesClusters[z][j][k], &dAreaCheck, iFlynnCluster );
1556              if ( dArea == dAreaCheck ) {
1557                vSplitClusterFlynns.push_back( vPhasesClusters[z][j][k] );
1558              }
1559            }
1560            if ( vSplitClusterFlynns.size() > 0 ) {
1561              vSplitClusters.push_back( vSplitClusterFlynns );
1562            }
1563          }
1564          // Wenn mehr als ein Cluster mit Flynns mit der gleichen Fläche gefunden wurde --> Der Cluster hat sich geteilt -->
1565    Flächen neu verteilen.
1566          // (Ein Cluster bedeutet der Cluster selbst wurde gefunden)
1567          if ( vSplitClusters.size() > 1 ) {
1568            //cout << "Cluster with same areanumber detected.... (SPLIT)" << endl;
1569            //for ( int g = 0; g < vSplitClusters.size(); g++ ) {
1570              //for ( int h = 0; h < vSplitClusters[g].size(); h++ ) {
1571                //cout << vSplitClusters[g][h] << " ";
1572              //}
1573              //cout << endl;
1574            //}
1575            clusterTracking::resolveSplit( vSplitClusters );
1576          }
1577        }
1578      }
1579    }
1580
1581    void clusterTracking::findClusters( void )
1582    {
1583      vPhasesClusterAreas.clear();
1584      vPhasesClusters.clear();
1585      vFlynns.clear();
1586      vFlynnPhase.clear();
1587
1588      int temp_int;
1589      double temp_double;
1590
1591      vector<int> ran;
1592
1593      iMaxFlynns = ElleMaxFlynns();
1594      ran.clear();
1595      for ( int i = 0; i < iMaxFlynns; i++ )
1596        if ( ElleFlynnIsActive( i ) )
1597          ran.push_back( i );
1598
1599      iMaxFlynns = ran.size();
1600
1601      for ( int i = 0; i < iMaxFlynns; i++ ) {
1602        vFlynns.push_back( ran.at( i ) );
1603        ElleGetFlynnRealAttribute( vFlynns.back(), &temp_double, iFlynnPhase );
1604        temp_int = (int)temp_double;
1605        vFlynnPhase.push_back( temp_int );
1606      }
1607
1608      if ( vFlynnPhase.size() != vFlynns.size() )
1609        cout << "ERROR: findClusters --> FlynnPhase and Flynn Vector don't have the same size!!!" << endl;
1610
1611      clusterTracking::getClusters();
```

```
1612      clusterTracking::getClusterAreas();
1613    }
1614
1615    void clusterTracking::findMerge( void )
1616    {
1617      double dArea, dAreaCheck;
1618
1619      list<double> lNotMatchingAreas;
1620
1621      for ( int z = 0; z < vPhasesClusters.size(); z++ ) {
1622        for ( int i = 0; i < vPhasesClusters[z].size(); i++ ) {
1623          lNotMatchingAreas.clear();
1624          ElleGetFlynnRealAttribute( vPhasesClusters[z][i][0], &dArea, iFlynnCluster );
1625          for ( int j = 1; j < vPhasesClusters[z][i].size(); j++ ) {
1626            ElleGetFlynnRealAttribute( vPhasesClusters[z][i][j], &dAreaCheck, iFlynnCluster );
1627            // if the Cluster Areas of the Flynn ain't match the first one... --> Merge?
1628            if ( dArea != dAreaCheck ) {
1629              lNotMatchingAreas.push_back(dAreaCheck);
1630            }
1631          }
1632          // all double Entries have to be deleted... --> problem if two of the merged clusters had the same areas...
1633          lNotMatchingAreas.sort();
1634          lNotMatchingAreas.unique();
1635
1636          if ( lNotMatchingAreas.size() > 0 ) {
1637            lNotMatchingAreas.push_back(dArea);
1638            clusterTracking::resolveMerge( z, i, lNotMatchingAreas );
1639          }
1640        }
1641      }
1642    }
1643
1644    void clusterTracking::checkDoubleClusterAreaLoop ( void )
1645    {
1646      bool bChange;
1647
1648      do {
1649        bChange = false;
1650        for ( int z = 0; z < vPhasesClusters.size() && !bChange; z++ )
1651          for ( int i = 0; i < vPhasesClusters[z].size() - 1 && !bChange; i++ )
1652            bChange = clusterTracking::checkDoubleClusterArea ( z, i, vPhasesClusters[z].size() - 1 );
1653      } while ( bChange == true );
1654    }
1655
1656    bool clusterTracking::checkDoubleClusterArea ( int z, int i, int iMax )
1657    {
1658      bool bChanged = false;
1659
1660      if ( i < iMax) {
1661        double dCheckA, dCheckB;
1662        ElleGetFlynnRealAttribute( vPhasesClusters[z][i][0], &dCheckA, iFlynnCluster );
1663        ElleGetFlynnRealAttribute( vPhasesClusters[z][iMax][0], &dCheckB, iFlynnCluster );
1664        if ( dCheckA == dCheckB ) {
1665          cout << "DOUBLE AREA DETECTED!!!" << endl;
1666          cout << dCheckA << " | Flynn: " << vPhasesClusters[z][i][0] << " (" << i << ") || " << dCheckB << " | Flynn: " <<
1667    vPhasesClusters[z][iMax][0] << " (" << iMax << ")" << endl;
1668          bChanged = clusterTracking::resolveDoubleClusterArea ( z, i, iMax, dCheckA );
1669        }
1670        if ( clusterTracking::checkDoubleClusterArea ( z, i, iMax-1 ) )
1671          bChanged = true;
1672      }
1673      return bChanged;
1674    }
1675
1676    bool clusterTracking::resolveDoubleClusterArea ( int z, int i, int iMax, double dCheck )
1677    {
1678      for ( int j = 0; j < vPhasesClusters[z][i].size(); j++ )
1679        ElleSetFlynnRealAttribute( vPhasesClusters[z][i][j], dCheck-dAreaShift, iFlynnCluster );
1680      for ( int j = 0; j < vPhasesClusters[z][iMax].size(); j++ )
1681        ElleSetFlynnRealAttribute( vPhasesClusters[z][iMax][j], dCheck+dAreaShift, iFlynnCluster );
1682
1683      return true;
1684    }
1685
1686    void clusterTracking::resolveMerge ( int z, int i, list<double> lNotMatchingAreas )
1687    {
1688      double dMergedArea = 0.0;
1689      // calculate new cluster area (just add the old areas together)
1690
1691      cout << "Cluster has different Clusterflynns... (MERGE) ";
1692      while ( lNotMatchingAreas.size() > 0) {
1693        cout << lNotMatchingAreas.back() << " ";
1694        dMergedArea += lNotMatchingAreas.back();
1695        lNotMatchingAreas.pop_back();
1696      }
1697      cout << ":: " << dMergedArea << endl;
1698      // set new area for ALL flynns in that cluster!
1699      for ( int j = 0; j < vPhasesClusters[z][i].size(); j++ ) {
1700        ElleSetFlynnRealAttribute( vPhasesClusters[z][i][j], dMergedArea, iFlynnCluster );
1701      }
1702    }
1703
1704    void clusterTracking::resolveSplit( vector<vector<int> > vSplitClusters )
1705    {
1706      double dSplitClusterAreas[vSplitClusters.size()], dSplitClusterRatio, dSplitClusterAreaComplete, dSplitClusterNewArea;
1707
1708      ElleGetFlynnRealAttribute( vSplitClusters[0][0], &dSplitClusterNewArea, iFlynnCluster );
1709      cout << "Cluster with same areanumber detected.... (SPLIT) " << dSplitClusterNewArea << " |";
1710      dSplitClusterAreaComplete = 0;
1711      for ( int j = 0; j < vSplitClusters.size(); j++ ) {
1712        dSplitClusterAreas[j] = 0;
```

```
    cout << "| ";
    for ( int k = 0; k < vSplitClusters[j].size(); k++ ) {
      dSplitClusterAreas[ j ] += ElleRegionArea( vSplitClusters[j][k] );
      cout << vSplitClusters[j][k] << " ";
    }
    dSplitClusterAreaComplete += dSplitClusterAreas[ j ];
  }
  cout << "|" << endl;
  for ( int j = 0; j < vSplitClusters.size(); j++ ) {
    // Calculate Ratio for that part of the split Cluster (Split Part / Current Complete Area) --> For the Ratio calculation
the old Area is not used.
    dSplitClusterRatio = dSplitClusterAreas[ j ] / dSplitClusterAreaComplete;
    // Calculate New Area with the OLD Area and the calculated Ratio
    ElleGetFlynnRealAttribute( vSplitClusters[j][0], &dSplitClusterNewArea, iFlynnCluster );
    dSplitClusterNewArea *= dSplitClusterRatio;
    // Write new Area in that part of the Flynn.
    for ( int k = 0; k < vSplitClusters[j].size(); k++ ) {
      ElleSetFlynnRealAttribute( vSplitClusters[j][k], dSplitClusterNewArea, iFlynnCluster );
    }
  }
}

bool fileExists(const char *filename)
{
  ifstream ifile(filename);
  return ifile;
}

int diffusearea(int mode, int max)
{
  int bracket = 20; // Used for inf and clust diffusion. Defines the bracket for the node amount in each cluster. 10 Means for
-9 to +9 nodes compared to the previous step the node amount of the previous step is used. A larger changes results in the
use of the current node amount.
  int  n, i, j, m, x=1, err, temp_int, check, found; //  infinite=0, infinite_p[3], *nodes, node_type, node,
  int nghbr[3], rgn[3], nn_rgn[3], type_i[3], nn_type_i[3], nnode[3], diff_nodes_size=0; //Diffusion stuff
  double type_a[3], nn_type_a[3], phase_area[phases.no_phases], temp_double, area;
  //int *inf_diff_p, infn, *clust_diff_p, clustn; // collection arrays and counter for infinite und cluster diffusion phases
  int on_list=0, *flynnnodes, flynnnodecount=0; //clusterdiffusion stuff

  vector<int> cluster;
  list<int> original, neighbour, doubles, triples, doubles_temp, triples_temp, infinite;
  list<int> inf_diff_p, clust_diff_p, diff_p, all_p;

  // copy the stuff from the overall list.
  inf_diff_p = lInfDiffPhases;
  clust_diff_p = lClustDiffPhases;
  diff_p = lFicksDiffPhases;
  all_p = lAllPhases;

  DiffNodes *diff_nodes;

  infinite.clear();

  // if there are cluster diffusion phases do cluster diffusion
  while (clust_diff_p.size()>0) {
    original.clear();
    // get all flynns with phase i
    for (j=0;j<max;j++) {
      if (grains[j].phase==clust_diff_p.front())
        original.push_back(grains[j].flynn);
    }
    // find flynns that are clustered together
    // as long as there are any flynns in the phase list
    while (original.size()>0) {
      cluster.clear();
      cluster.push_back(original.front()); // put the first flynn into the cluster list
      original.pop_front(); // delete that element from the list
      for (n=0;n<cluster.size();n++) {
        neighbour.clear(); // clear the neigbour list
        ElleFlynnNbRegions(cluster.at(n), neighbour); //find neighbours for the current flynn (n) in the cluster list

        //check whether any flynn in the neighbour list matches the current phase (i)
        // as long as there are entries in the neighbours list do the following
        while (neighbour.size()>0) {
          ElleGetFlynnRealAttribute(neighbour.front(), &temp_double, iFlynnPhase); // get phase from flynn
          temp_int = (int)temp_double; // convert to int

          //compare to current phase
          if (temp_int == clust_diff_p.front()) {                    // if right
            for (i=0,on_list=0;i<cluster.size() && on_list==0;i++) // look whether the Flynn is already on the cluster list
              if (neighbour.front()==cluster.at(i))
                on_list=1;
            if (on_list==0) {                                                    // if NOT
              cluster.push_back(neighbour.front()); // add flynn to cluster list
              original.remove(neighbour.front());            // remove that flynn from the original phase list
              neighbour.pop_front();                                          // remove it from the neighbours list
            }
            else                                                                 // if it is
              neighbour.pop_front();                                          // just remove it from the neighbours list
          }
          else                                                                 // if not right
            neighbour.pop_front();                                          // just remove it from the neighbours list
        }
      }

      // for every flynn on the cluster list -> find the nodes and separate them on a list for doubles and triples.
      doubles_temp.clear();
      triples_temp.clear();
      while (cluster.size()>0) {
        ElleFlynnNodes(cluster.back(), &flynnnodes, &flynnnodecount);
        for (j=0;j<flynnnodecount;j++) {
```

```
1814            if (ElleNodeIsDouble(*(flynnnodes+j)))
1815              doubles_temp.push_back(*(flynnnodes+j));
1816            else if (ElleNodeIsTriple(*(flynnnodes+j)))
1817              triples_temp.push_back(*(flynnnodes+j));
1818            else
1819              printf("ERROR: diffusearea: node.push_back\n");
1820          }
1821          cluster.pop_back();
1822          free(flynnnodes);
1823          flynnnodecount=0;
1824
1825        }
1826        // sort the two lists
1827        doubles_temp.sort();
1828        triples_temp.sort();
1829        doubles.clear();
1830        triples.clear();
1831
1832        // clear the doubles list of double entries
1833        // those are the doubles nodes enclosed in the same phase and therefore don't matter for diffusion
1834        temp_int=-1;
1835        while (doubles_temp.size()>0) {
1836          temp_int=doubles_temp.front();                                  // first one
1837          doubles_temp.pop_front();                                               // delete "first" list entry
1838          if (temp_int==doubles_temp.front()) {               // compare to the "second" entry
1839            doubles_temp.remove(temp_int);                    // if the same, delete all entries with that number
1840          }
1841          else {
1842            doubles_temp.remove(temp_int);                    // if not the same as the "second" delete all entries with that
1843    number, but
1844            doubles.push_back(temp_int);             // add it to the end of the list again
1845          }
1846        }
1847
1848        // clear the triples list of triple entries (double entries are still valid)
1849        temp_int=-1;
1850        while (triples_temp.size()>0) {
1851          temp_int=triples_temp.front();                                  // first one
1852          triples_temp.pop_front();                                               // delete first list entry
1853          if (temp_int==triples_temp.front()) {               // compare to the first entry (actually the second, because the
1854    first one got deleted)
1855            triples_temp.pop_front();                                     // if the same delete the "second" entry
1856            if (temp_int==triples_temp.front())               // compare to the "third" entry
1857              triples_temp.remove(temp_int);                  // if the same, delete all entries with that number
1858            else {
1859              triples_temp.remove(temp_int);                  // if not the same as the "third" delete all entries, but
1860              triples.push_back(temp_int);  // add it to the end of the list again
1861            }
1862          }
1863          else {
1864            triples_temp.remove(temp_int);                    // if not the same as the "second" delete all entries, but
1865            triples.push_back(temp_int);             // add it to the end of the list again
1866          }
1867        }
1868
1869        // just in case delete all double entries of the lists
1870        triples.unique();
1871        doubles.unique();
1872
1873        //combine both lists
1874        doubles.splice (doubles.begin(), triples);
1875
1876        // CHECK whether there are more or less boundary nodes along a cluster
1877        int node_amount = (int) ElleNodeAttribute(doubles.front(), iClusterNodeCount);
1878        int node_amount_new = (int) doubles.size();
1879
1880        for (j=0;j<phases.no_phases;j++) {
1881          // summarize all areas from the nodes of that phase
1882          for (i=0,temp_double=0;i<doubles.size();i++) {
1883            temp_double += ElleNodeAttribute(doubles.front(), attrib[j]);
1884            doubles.push_back(doubles.front());
1885            doubles.pop_front();
1886          }
1887
1888    //      cout << "Cluster: " << original.size() << " Phase: " << j << " AreaEnergy: " << scientific << temp_double << endl;
1889
1890          if ( node_amount < 0 )
1891            cout << "Invalid node amount...." << endl;
1892          else if ( node_amount == 0 ) {
1893            temp_double /= doubles.size(); // Probably for the first step. Has to be extra case otherwise division by 0
1894    possible....
1895            // cout << "There would have been an impossible division at timestep: " << Settings_run.Count << " at Node: " <<
1896    doubles.front() << endl;
1897          }
1898          else if ( node_amount > node_amount_new - bracket && node_amount < node_amount_new + bracket )
1899            temp_double /= node_amount;              // divide area equally however use the previous node count in case there are
1900    nodes lost/added in the last time step.
1901          else
1902            temp_double /= doubles.size();  // divide the area equally
1903
1904          // set the new area for all nodes
1905          for (i=0;i<doubles.size();i++) {
1906            if ( j > 0 && doubles.size() != (int) ElleNodeAttribute(doubles.front(), iClusterNodeCount))
1907              cout << "Node Amount Error" << endl;
1908            ElleSetNodeAttribute(doubles.front(), (double) node_amount_new, iClusterNodeCount);// set the node amount for this
1909    time step for the nodes as well.
1910            ElleSetNodeAttribute(doubles.front(), temp_double, attrib[j]);
1911            doubles.push_back(doubles.front());
1912            infinite.push_back(doubles.front()); //add all the cluster nodes to a list which will be later on removed from the
1913    all nodes list
1914            doubles.pop_front();
```

```
1915            if (mode==2) {
1916              fp=fopen("diff_cluster.txt", "a");
1917              fprintf(fp,"%d\t%le\t%d\t%d\n",doubles.back(),temp_double,original.size(),j);
1918              fclose(fp);
1919            }
1920          }
1921        }
1922
1923    // OUTDATED --> New code works for all phases that lie on a infinite boundary and not just that one phase...
1924    //    // summarize all areas from the nodes of that phase
1925    //    for (i=0,temp_double=0;i<doubles.size();i++) {
1926    //      temp_double += ElleNodeAttribute(doubles.front(), attrib[clust_diff_p.front()]);
1927    //      doubles.push_back(doubles.front());
1928    //      doubles.pop_front();
1929    //    }
1930    //
1931    //    // divide the area equally
1932    //    temp_double /= doubles.size();
1933    //
1934    //    // set the new area for all nodes
1935    //    for (i=0;i<doubles.size();i++) {
1936    //      ElleSetNodeAttribute(doubles.front(), temp_double, attrib[clust_diff_p.front()]);
1937    //      doubles.push_back(doubles.front());
1938    //      infinite.push_back(doubles.front()); //add all the cluster nodes to a list which will be later on removed from the
1939    all nodes list
1940    //      doubles.pop_front();
1941    //      if (mode==2) {
1942    //        fp=fopen("diff_cluster.txt", "a");
1943    //        fprintf(fp,"%d\t%le\t%d\n",doubles.back(),temp_double,original.size());
1944    //        fclose(fp);
1945    //      }
1946    //    }
1947
1948
1949        }
1950    //   inf_diff_p.remove(clust_diff_p.front()); //not needed anymore since it changed a bit
1951        clust_diff_p.pop_front();
1952
1953
1954      }
1955      // cluster diffusion finished
1956      //#############################################################################
1957      //#############################################################################
1958      //#############################################################################
1959      // INFINTE DIFFUSION
1960      //basicly the same as cluster diffusion without the finding of clusters at the beginning
1961      //just add all the nodes of all the flynns that have this phase and remove the double double_nodes and triple triple_nodes
1962
1963      while (inf_diff_p.size()>0) {
1964        original.clear();
1965        // get all flynns with phase i
1966        for (j=0;j<max;j++) {
1967          if (grains[j].phase==inf_diff_p.front())
1968            original.push_back(grains[j].flynn);
1969        }
1970        // find flynns that are clustered together
1971        // as long as there are any flynns in the phase list
1972        doubles_temp.clear();
1973        triples_temp.clear();
1974        while (original.size()>0) {
1975
1976          ElleFlynnNodes(original.back(), &flynnnodes, &flynnnodecount);
1977          for (j=0;j<flynnnodecount;j++) {
1978            if (ElleNodeIsDouble(*(flynnnodes+j)))
1979              doubles_temp.push_back(*(flynnnodes+j));
1980            else if (ElleNodeIsTriple(*(flynnnodes+j)))
1981              triples_temp.push_back(*(flynnnodes+j));
1982            else
1983              printf("ERROR: diffusearea: inf_diff: node.push_back\n");
1984          }
1985        original.pop_back();
1986        free(flynnnodes);
1987        flynnnodecount=0;
1988
1989        }
1990        // sort the two lists
1991        doubles_temp.sort();
1992        triples_temp.sort();
1993        doubles.clear();
1994        triples.clear();
1995
1996        // clear the doubles list of double entries
1997        // those are the doubles nodes enclosed in the same phase and therefore don't matter for diffusion
1998        temp_int=-1;
1999        while (doubles_temp.size()>0) {
2000          temp_int=doubles_temp.front();                                    // first one
2001          doubles_temp.pop_front();                                                 // delete "first" list entry
2002          if (temp_int==doubles_temp.front()) {                 // compare to the "second" entry
2003            doubles_temp.remove(temp_int);                      // if the same, delete all entries with that number
2004          }
2005          else {
2006            doubles_temp.remove(temp_int);                      // if not the same as the "second" delete all entries with that
2007    number, but
2008            doubles.push_back(temp_int);              // add it to the end of the list again
2009          }
2010        }
2011
2012        // clear the triples list of triple entries (double entries are still valid)
2013        temp_int=-1;
2014        while (triples_temp.size()>0) {
2015          temp_int=triples_temp.front();                                    // first one
```

```
2016         triples_temp.pop_front();                                                  // delete first list entry
2017         if (temp_int==triples_temp.front()) {                 // compare to the first entry (actually the second, because the
2018     first one got deleted)
2019           triples_temp.pop_front();                                        // if the same delete the "second" entry
2020           if (temp_int==triples_temp.front())                 // compare to the "third" entry
2021             triples_temp.remove(temp_int);           // if the same, delete all entries with that number
2022           else {
2023             triples_temp.remove(temp_int);           // if not the same as the "third" delete all entries, but
2024             triples.push_back(temp_int);    // add it to the end of the list again
2025           }
2026         }
2027         else {
2028           triples_temp.remove(temp_int);                     // if not the same as the "second" delete all entries, but
2029           triples.push_back(temp_int);              // add it to the end of the list again
2030         }
2031       }
2032
2033       // just in case delete all double entries of the lists
2034       triples.unique();
2035       doubles.unique();
2036
2037       //combine both lists
2038       doubles.splice (doubles.begin(), triples);
2039
2040       // CHECK whether there are more or less boundary nodes along a cluster
2041       int node_amount = 0; (int) ElleNodeAttribute(doubles.front(), iClusterNodeCount);
2042       int node_amount_new = doubles.size();
2043
2044       for (j=0;j<phases.no_phases;j++) {
2045         // summarize all areas from the nodes of that phase
2046         for (i=0,temp_double=0;i<doubles.size();i++) {
2047           temp_double += ElleNodeAttribute(doubles.front(), attrib[j]);
2048           node_amount += (int) ElleNodeAttribute(doubles.front(), iClusterNodeCount);
2049           doubles.push_back(doubles.front());
2050           doubles.pop_front();
2051         }
2052
2053         node_amount /= node_amount_new;
2054
2055         if ( node_amount < 0 )
2056           cout << "Invalid node amount...." << endl;
2057         else if ( node_amount == 0 )
2058           temp_double /= doubles.size(); // Probably for the first step. Has to be extra case otherwise division by 0
2059     possible....
2060         else if ( node_amount > doubles.size()-bracket && node_amount < doubles.size()+bracket )
2061           temp_double /= node_amount;                // divide area equally however use the previous node count in case there are
2062     nodes lost/added in the last time step.
2063         else
2064           temp_double /= doubles.size();   // divide the area equally
2065
2066         // set the new area for all nodes
2067         for (i=0;i<doubles.size();i++) {
2068           if ( j > 0 && doubles.size() != (int) ElleNodeAttribute(doubles.front(), iClusterNodeCount))
2069             cout << "Node Amount Error" << endl;
2070           ElleSetNodeAttribute(doubles.front(), (double) node_amount_new, iClusterNodeCount);// set the node amount for this time
2071     step for the nodes as well.
2072           ElleSetNodeAttribute(doubles.front(), temp_double, attrib[j]);
2073           doubles.push_back(doubles.front());
2074           doubles.pop_front();
2075           if (mode==2) {
2076             fp=fopen("diff_infinite.txt", "a");
2077             fprintf(fp,"%d\t%le\t%d\n",doubles.back(),temp_double, j);
2078             fclose(fp);
2079           }
2080         }
2081       }
2082
2083     //   // summarize all areas from the nodes of that phase
2084     //   for (i=0,temp_double=0;i<doubles.size();i++) {
2085     //     temp_double += ElleNodeAttribute(doubles.front(), attrib[inf_diff_p.front()]);
2086     //     doubles.push_back(doubles.front());
2087     //     doubles.pop_front();
2088     //   }
2089     //
2090     //   // divide the area equally
2091     //   temp_double /= doubles.size();
2092     //
2093     //   // set the new area for all nodes
2094     //   for (i=0;i<doubles.size();i++) {
2095     //     ElleSetNodeAttribute(doubles.front(), temp_double, attrib[inf_diff_p.front()]);
2096     //     doubles.push_back(doubles.front());
2097     //     doubles.pop_front();
2098     //     if (mode==2) {
2099     //       fp=fopen("diff_infinite.txt", "a");
2100     //       fprintf(fp,"%d\t%le\n",doubles.back(),temp_double);
2101     //       fclose(fp);
2102     //     }
2103     //   }
2104
2105       inf_diff_p.pop_front();
2106
2107      }
2108
2109      // infinite diffusion finished
2110      //#############################################################################
2111      //#############################################################################
2112      //#############################################################################
2113      // DIFFUSION
2114
2115      // find all nodes
2116      max = ElleMaxNodes();
```

```
2117      doubles.clear();
2118      triples.clear();
2119      doubles_temp.clear();
2120      triples_temp.clear();
2121
2122
2123      for (i=0;i<max;i++) {
2124        if (ElleNodeIsActive(i)) {
2125          if (ElleNodeIsDouble(i))
2126            doubles.push_back(i);
2127          else if (ElleNodeIsTriple(i))
2128            triples.push_back(i);
2129          else
2130            printf("ERROR: this is not possible yet\n");
2131        }
2132      }
2133
2134      //************************
2135      //Added to delete the nodes which already diffused infinite from the list
2136      doubles.sort();
2137      triples.sort();
2138      infinite.sort();
2139      infinite.unique();
2140      while (infinite.size()>0) {
2141        doubles.remove(infinite.front());
2142        triples.remove(infinite.front());
2143        infinite.pop_front();
2144      }
2145      //don't think this is necessary...
2146     // random_shuffle(doubles.begin(),doubles.end());
2147     // random_shuffle(triples.begin(),triples.end());
2148      //************************
2149
2150      if ((diff_nodes = (DiffNodes *)malloc((2*doubles.size()+3*triples.size())*sizeof(DiffNodes)))== 0)
2151        printf("ERROR: diffusearea: Malloc_Err: diff_nodes\n");
2152
2153
2154
2155
2156
2157      //reset the counter
2158      m=0;
2159
2160
2161      //*****************************DOUBLE NODES*****************************
2162      while (doubles.size()>0) {
2163
2164
2165
2166        // find the neighbours
2167        if (err=ElleNeighbourNodes(doubles.front(),nghbr))
2168          printf("ERROR: diffusearea: doublenode diffusion");
2169        // read the attributes of that node
2170        for (n=0,j=0;n<3;n++)
2171        {
2172          if(nghbr[n]!=NO_NB && j<2) {
2173            nnode[j] = nghbr[n];
2174            ElleNeighbourRegion(doubles.front(),nghbr[n],&rgn[j]);
2175            ElleGetFlynnRealAttribute(rgn[j], &type_a[j], iFlynnPhase);
2176            type_i[j] = (int)type_a[j];
2177            j++;
2178          }
2179        }
2180        // put the stuff into the array
2181        if (type_i[0]==type_i[1]) {
2182          diff_nodes[m].node=doubles.front();
2183          diff_nodes[m].not_diff=1;                           // 2 phases are the same
2184          diff_nodes[m].newconc=0.0;
2185          //needed for shiftarea
2186          diff_nodes[m].nb1=nnode[0];
2187          diff_nodes[m].nb2=nnode[1];
2188          //inserted just for readability of the logfile...
2189          diff_nodes[m].p=0;
2190          diff_nodes[m].nb1_p1=0;
2191          diff_nodes[m].nb1_p2=0;
2192          diff_nodes[m].nb2_p1=0;
2193          diff_nodes[m].nb2_p2=0;
2194          m++;
2195        }
2196        else {                                        // are not the same
2197          for (i=0;i<2;i++) {
2198            diff_nodes[m].node=doubles.front();
2199            diff_nodes[m].p=type_i[i];
2200            diff_nodes[m].nb1=nnode[0];
2201            diff_nodes[m].nb2=nnode[1];
2202            diff_nodes[m].nb1_p1=type_i[0];
2203            diff_nodes[m].nb1_p2=type_i[1];
2204            diff_nodes[m].nb2_p1=type_i[0];
2205            diff_nodes[m].nb2_p2=type_i[1];
2206            diff_nodes[m].not_diff=2;
2207            diff_nodes[m].newconc=0.0;
2208            m++;
2209          }
2210        }
2211        // delete the node from the double list
2212
2213        doubles.pop_front();
2214      }
2215
2216      //*****************************TRIPLE NODES*****************************
2217
```

```
  while (triples.size()>0) {


    // find the neighbours
    if (err=ElleNeighbourNodes(triples.front(),nghbr))
      OnError("diffusearea: triplenode diffusion",err);
    // read the attributes of that node
    for (n=0,j=0;n<3;n++)
    {
      if(nghbr[n]!=NO_NB && j<3) {
        nnode[j] = nghbr[n];
        //from node to neighbour
        ElleNeighbourRegion(triples.front(),nnode[j],&rgn[j]);
        ElleGetFlynnRealAttribute(rgn[j], &type_a[j], iFlynnPhase);
        type_i[j] = (int)type_a[j];
        //from neighbour to node
        ElleNeighbourRegion(nnode[j],triples.front(),&nn_rgn[j]);
        ElleGetFlynnRealAttribute(nn_rgn[j], &nn_type_a[j], iFlynnPhase);
        nn_type_i[j] = (int)nn_type_a[j];
        j++;
      }
    }
    if (type_i[0]==type_i[1]) {
      if (type_i[0]==type_i[2]) {
        diff_nodes[m].node=triples.front();
        diff_nodes[m].not_diff=1;                  // every 3 phases are the same
        diff_nodes[m].newconc=0.0;
        //add at least a triple node to the list -- needed for shiftarea
        if (ElleNodeIsTriple(nnode[0]))
          diff_nodes[m].nb1=nnode[0];
        else
          diff_nodes[m].nb1=nnode[1];
        diff_nodes[m].nb2=nnode[2];
        //inserted just for readability of the logfile...
        diff_nodes[m].p=0;
        diff_nodes[m].nb1_p1=0;
        diff_nodes[m].nb1_p2=0;
        diff_nodes[m].nb2_p1=0;
        diff_nodes[m].nb2_p2=0;
        m++;
      }
      else {                                       // phase 2 is different from 0 and 1
        for (i=1;i<3;i++) {                                 // 2 entries for phase 0 and 2, since 0 and 1 are the same it
could also be from 1 till 2.
          diff_nodes[m].node=triples.front();
          diff_nodes[m].p=type_i[i];
          diff_nodes[m].nb1=nnode[0];
          diff_nodes[m].nb2=nnode[2];
          diff_nodes[m].nb1_p1=type_i[0];
          diff_nodes[m].nb1_p2=nn_type_i[0];
          diff_nodes[m].nb2_p1=type_i[2];
          diff_nodes[m].nb2_p2=nn_type_i[2];
          diff_nodes[m].not_diff=3;
          diff_nodes[m].newconc=0.0;
          m++;
        }
      }
    }
    else {
      if (type_i[0]==type_i[2]) {                           // phase 1 is different from 0 and 2
        for (i=1;i<3;i++) {                                 // 2 entries for phase 1 and 2->(same as 0)
          diff_nodes[m].node=triples.front();
          diff_nodes[m].p=type_i[i];
          diff_nodes[m].nb1=nnode[1];
          diff_nodes[m].nb2=nnode[2];
          diff_nodes[m].nb1_p1=type_i[1];
          diff_nodes[m].nb1_p2=nn_type_i[1];
          diff_nodes[m].nb2_p1=type_i[2];
          diff_nodes[m].nb2_p2=nn_type_i[2];
          diff_nodes[m].not_diff=3;
          diff_nodes[m].newconc=0.0;
          m++;
        }
      }
      else if (type_i[1]==type_i[2]) {   // phase 0 is different from 1 and 2
        for (i=0;i<2;i++) {                                 // 2 entries for phase 0 and 1 (same as 2)
          diff_nodes[m].node=triples.front();
          diff_nodes[m].p=type_i[i];
          diff_nodes[m].nb1=nnode[0];
          diff_nodes[m].nb2=nnode[1];
          diff_nodes[m].nb1_p1=type_i[0];
          diff_nodes[m].nb1_p2=nn_type_i[0];
          diff_nodes[m].nb2_p1=type_i[1];
          diff_nodes[m].nb2_p2=nn_type_i[1];
          diff_nodes[m].not_diff=3;
          diff_nodes[m].newconc=0.0;
          m++;
        }
      }
      else if (type_i[0]!=type_i[2] && type_i[1]!=type_i[2]) {        // every 3 phases different
        for (i=0;i<3;i++) {                                                                                 // 3
entries every phase
          diff_nodes[m].node=triples.front();
          diff_nodes[m].p=type_i[i];
          diff_nodes[m].nb1=nnode[i];
          for (j=0, found=0;j<3;j++) {
            if (nn_type_i[j] == type_i[i]) {
              diff_nodes[m].nb2=nnode[j];
              found=1;
            }
          }
```

```
2319            if (found==0)
2320              printf("ERROR: diffusearea: triple_node - 3phases: no 2nd neighbour found\n");
2321
2322            diff_nodes[m].nb1_p1=type_i[i];
2323            diff_nodes[m].nb1_p2=nn_type_i[i];
2324            diff_nodes[m].nb2_p1=type_i[j];
2325            diff_nodes[m].nb2_p2=nn_type_i[j];
2326            diff_nodes[m].not_diff=4;
2327            diff_nodes[m].newconc=0.0;
2328            m++;
2329           }
2330          }
2331        }
2332
2333        triples.pop_front();
2334      }
2335
2336      diff_nodes_size=m;
2337
2338
2339    // fp=fopen("log.txt", "a");
2340    // fprintf(fp,
2341    "##############################################\n##############################################\n######################
2342    ######################\n");
2343    // fprintf(fp, "\n\nNODE\tNB1\tNB2\tNB1_P1\tNB1_P2\tNB2_P1\tNB2_P2\tNOT_DIFF\tNEWCONC\n");
2344    // for (i=0;i<diff_nodes_size;i++)
2345    //   fprintf(fp, "%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%le\n", diff_nodes[i].node, diff_nodes[i].nb1, diff_nodes[i].nb2,
2346    diff_nodes[i].nb1_p1, diff_nodes[i].nb1_p2, diff_nodes[i].nb2_p1, diff_nodes[i].nb2_p2, diff_nodes[i].not_diff,
2347    diff_nodes[i].newconc);
2348    // fclose(fp);
2349
2350
2351      //do shiftarea
2352      while (all_p.size()>0) {
2353        for (i=0;i<diff_nodes_size;i++) {
2354
2355          // if the node is surrounded by one phase --> check if there is area of the current phase to diffuse back to the
2356    boundary.
2357          // but only one time, for each step and phase. For additional diffusion_times from the config file there is no need for
2358    shiftarea
2359          if (diff_nodes[i].not_diff==1) {
2360            if ((area=ElleNodeAttribute(diff_nodes[i].node, attrib[all_p.front()]))!=0.0) {
2361              //printf("Node: %d --> area:%le\n", diff_nodes[i].node, area);
2362              x=shiftarea(diff_nodes[i].node, diff_nodes[i].nb1, diff_nodes[i].nb2, all_p.front());
2363    //          if (x==0)
2364    //            printf("ERROR: diffusearea: shiftarea: node: %d\n", diff_nodes[i].node);
2365            }
2366          }
2367        }
2368        all_p.pop_front();
2369      }
2370
2371      // as long as there are phases to diffuse
2372      while (diff_p.size()>0) {
2373
2374        // diffuse for x diffusion steps in the config file
2375        for (j=0;j<phases.phasep[diff_p.front()].diffusion_times;j++) {
2376
2377          // for all the nodes
2378          for (i=0;i<diff_nodes_size;i++) {
2379
2380            if (diff_nodes[i].not_diff>1) {
2381              x=diffuse_dn(i, diff_nodes);
2382              if (x==0)
2383                printf("ERROR: diffusearea: diffuse_dn: node: %d\n", diff_nodes[i].node);
2384            }
2385    //        fprintf(fp, "%le\n", diff_nodes[i].newconc);
2386            // Just to print the areas for the last diff step for all nodes
2387            if (mode==2) {
2388              if (j==((phases.phasep[diff_p.front()].diffusion_times)-1)) {
2389                fp=fopen("diff_normal.txt", "a");
2390                fprintf(fp,"%d\t%le\t%d\n",diff_nodes[i].node,diff_nodes[i].newconc,diff_nodes[i].not_diff);
2391                fclose(fp);
2392              }
2393            }
2394
2395
2396          } // end of all nodes
2397    //      fprintf(fp, "\n");
2398    //      fprintf(fp, "node %d conc %le\n", diff_nodes[1].node, diff_nodes[1].newconc);
2399    //      fprintf(fp, "Size: %d\n", diff_nodes_size);
2400    //      fclose(fp);
2401          writenewconc(diff_nodes_size, diff_nodes);
2402        } // end of diffusion_times
2403
2404        // delete the diffusion phase from the list
2405        diff_p.pop_front();
2406      } // end of diff_p
2407
2408    // for (i=0;i<max;i++)
2409    //   printf("flynn-phase: %d-%d\n", grains[i].flynn, grains[i].phase);
2410
2411
2412      //free malloc arrays
2413      free(diff_nodes);
2414      diff_nodes = NULL;
2415
2416      //x=1;
2417      return (x);
2418    }
2419
```

```
2420  int shiftarea(int node, int n1, int n2, int type)
2421  {
2422    int found=-1, direction=0, nnode[3], nghbr[3], rgn[3], type_i[3], current, next, last, x, n, j, i;
2423    double area, old_area, type_a[3];
2424
2425    for (i=0;i<2 && found<0;i++) {
2426      last=node;
2427      if (i==0)
2428        current=n1;
2429      else
2430        current=n2;
2431
2432      while (found<0 && direction==i) { // && ElleNodeAttribute(n1, attrib[type])>0.0)
2433        // find the neighbours
2434        if (x=ElleNeighbourNodes(current,nghbr))
2435          printf("ERROR: shiftarea: find neighbours for node: %d\n", current); //OnError("diffusearea: triplenode
2436  diffusion",err);
2437       // check if node is triple
2438        if (ElleNodeIsTriple(current)==1) {
2439          // read the attributes of that node
2440          for (n=0;n<3 && found<0;n++) {
2441            ElleNeighbourRegion(current,nghbr[n],&rgn[n]);
2442            ElleGetFlynnRealAttribute(rgn[n], &type_a[n], iFlynnPhase);
2443            type_i[n] = (int)type_a[n];
2444          }
2445          for (n=0;n<3 && found<0;n++) {
2446            if (type_i[n]==type) {
2447              for (j=0;j<3 && found<0;j++) {
2448                if (type_i[j]!=type)
2449                  found=current;
2450              }
2451            }
2452          }
2453          if (found<0) {
2454            if (i==0)
2455              direction=1;
2456            else
2457              direction=4;
2458          }
2459        }
2460        // if it isn't triple, find the next node in that direction
2461        else if (ElleNodeIsDouble(current)==1){
2462          // find next node
2463          for (n=0;n<3;n++) {
2464            if(nghbr[n]!=NO_NB && nghbr[n]!=last)
2465              next=nghbr[n];
2466          }
2467          last=current;
2468          current=next;
2469        }
2470      }
2471    }
2472  // else
2473  //   printf("ERROR: shiftarea: no triple neighbour found for node: %d\n", node);
2474
2475
2476    // if a shiftnode was found the area of the old node is added to the area already on the shiftnode
2477    // everything only for phase=type
2478    if (found>=0) {
2479      //printf("found: %d\n", found);
2480      area = ElleNodeAttribute(node, attrib[type]);
2481      old_area = ElleNodeAttribute(found, attrib[type]);
2482      ElleSetNodeAttribute(node, 0.0, attrib[type]);
2483      area += old_area;
2484      ElleSetNodeAttribute(found, area, attrib[type]);
2485      return (1);
2486    }
2487    else
2488      return (0);
2489  }
2490
2491
2492  int diffuse_dn(int position, DiffNodes *nodes)
2493  {
2494
2495    int dt_int;
2496    double tmp1, tmp2, tmp3;
2497    double double_kappa, kappa, dt, temp_conc, temp_segwidth, temp_seglength, s;
2498
2499    kappa=phases.phasep[nodes[position].p].kappa;
2500
2501    double_kappa=kappa*2.0; // was kappa*2 but since I don'T know yet what kappa is.....
2502    dt = ((ElleTimestep()*ElleSpeedup())/((ElleSwitchdistance()*ElleSwitchdistance())/kappa*0.25));
2503    dt_int = (int)dt;
2504    if (dt_int<MIN_DIFF_DT)
2505      dt_int=MIN_DIFF_DT;
2506    dt = ElleTimestep()*ElleSpeedup()/dt_int;
2507
2508    tmp1 = tmp2 = tmp3 = 0;
2509
2510    // FIRST Neighbour
2511    temp_conc=ElleNodeAttribute(nodes[position].nb1, attrib[nodes[position].p]);
2512    temp_segwidth=ElleBndWidth()/ElleUnitLength();
2513    temp_seglength=ElleNodeSeparation(nodes[position].node,nodes[position].nb1);
2514
2515    s=(kappa*dt)/(temp_seglength*temp_seglength);
2516    if(s<=0 || s>=0.5)
2517      printf("ERROR: diffuse_dn: s is not in range for explicite finite difference!!! (1st neighbour)\n");
2518
2519
2520    tmp1 += temp_conc*temp_segwidth/temp_seglength;
```

```
2521      tmp2 += temp_segwidth*temp_seglength;
2522      tmp3 += temp_segwidth/temp_seglength;
2523
2524      // SECOND Neighbour
2525      temp_conc=ElleNodeAttribute(nodes[position].nb2, attrib[nodes[position].p]);
2526      temp_seglength=ElleNodeSeparation(nodes[position].node,nodes[position].nb2);
2527
2528      s=(kappa*dt)/(temp_seglength*temp_seglength);
2529      if(s<=0 || s>=0.5)
2530        printf("ERROR: diffuse_dn: s is not in range for explicite finite difference!!! (2nd neighbour)\n");
2531
2532
2533      tmp1 += temp_conc*temp_segwidth/temp_seglength;
2534      tmp2 += temp_segwidth*temp_seglength;
2535      tmp3 += temp_segwidth/temp_seglength;
2536
2537      // NODE itself
2538
2539      temp_conc=ElleNodeAttribute(nodes[position].node, attrib[nodes[position].p]);
2540
2541      nodes[position].newconc =  temp_conc*(1 - double_kappa*dt*(tmp3/tmp2)) + (double_kappa*dt*tmp1/tmp2);
2542
2543
2544      return (1);
2545    }
2546
2547    void writenewconc(int max, DiffNodes *nodes)
2548    {
2549      int i;
2550
2551      for (i=0;i<max;i++)
2552        ElleSetNodeAttribute(nodes[i].node, nodes[i].newconc, attrib[nodes[i].p]);
2553    }
2554
2555    void mergeair(int mode)
2556    {
2557      int  i, x=1, max, temp_int, found, trys;
2558      double temp_double;
2559
2560      vector<int> ran;
2561      list<int> original, neighbour;
2562      list<int> merge_p;
2563      Flynnies *flynns;
2564
2565
2566      // find all flynns
2567      max = ElleMaxFlynns();
2568      ran.clear();
2569      for (i=0;i<max;i++)
2570        if (ElleFlynnIsActive(i))
2571          ran.push_back(i);
2572      max=ran.size();
2573
2574      if ((flynns = (Flynnies *)malloc(max*sizeof(Flynnies)))== 0)
2575          printf("ERROR: diffusearea: Malloc_Err: flynns\n");
2576
2577      for (i=0;i<max;i++) {
2578        flynns[i].flynn=ran.at(i);
2579        ElleGetFlynnRealAttribute(flynns[i].flynn, &temp_double, iFlynnPhase);
2580        temp_int = (int)temp_double;
2581        flynns[i].phase=temp_int;
2582      }
2583
2584      // check which phases to merge
2585      merge_p.clear();
2586      for (i=0;i<phases.no_phases;i++) {
2587        if (phases.phasep[i].merge==1)
2588          merge_p.push_back(i);
2589      }
2590
2591      // if there are cluster diffusion phases do cluster diffusion
2592      while (merge_p.size()>0) {
2593        original.clear();
2594        // get all flynns with phase i
2595        for (i=0;i<max;i++) {
2596          if (flynns[i].phase==merge_p.front())
2597            original.push_back(flynns[i].flynn);
2598        }
2599        // find flynns that are clustered together
2600        // as long as there are any flynns in the phase list
2601        while (original.size()>0) {
2602          found=1;
2603          trys=0;
2604          while (found==1 && trys<5) { //for (n=0;n<cluster.size();n++) {
2605            found=0; // will be set 1 again if the loop finds neighbours of the same phase
2606            neighbour.clear(); // clear the neigbour list
2607            ElleFlynnNbRegions(original.front(), neighbour); //find neighbours for the current flynn (n) in the cluster list
2608
2609            //check whether any flynn in the neighbour list matches the current phase (i)
2610            // as long as there are entries in the neighbours list do the following
2611            while (neighbour.size()>0) {
2612              ElleGetFlynnRealAttribute(neighbour.front(), &temp_double, iFlynnPhase); // get phase from flynn
2613              temp_int = (int)temp_double; // convert to int
2614
2615              //compare to current phase
2616              if (temp_int == merge_p.front()) { // if right
2617                found=1;
2618                x=ElleMergeFlynnsNoCheck(original.front(), neighbour.front());
2619                if (x==0) {
2620                  original.remove(neighbour.front());
2621                  neighbour.pop_front();
```

```
2622              }
2623              else {
2624                printf("ERROR: mergeair: flynn merge error between flynn %d and %d\n", original.front(), neighbour.front());
2625                printf("No solution to that problem yet. Maybe it works later on...--> NOT MERGING\n");
2626                neighbour.pop_front();
2627                trys++;
2628              }
2629            }
2630            else                                                          // if not right
2631              neighbour.pop_front();
2632          }
2633        }
2634        original.pop_front();
2635      }
2636      merge_p.pop_front();
2637    }
2638    free(flynns);
2639  }
2640
2641  int savearea(int mode, int max)
2642  {
2643    int i, j, c, x, found=0;
2644    double area[phases.no_phases], org_area[phases.no_phases], diff;
2645
2646    FILE *af, *sh;
2647
2648    for (i=0;i<phases.no_phases;i++) {
2649      area[i]=0;
2650    }
2651
2652    for ( int i = 0; i < phases.no_phases; i++ ) {
2653      area[i]=0; // set the start to 0
2654      for ( int j = 0; j < max; j++ ) {
2655        if ( grains[j].phase == i && ElleFlynnIsActive(grains[j].flynn) ) {
2656          area[i] += ElleRegionArea(grains[j].flynn);
2657        }
2658      }
2659    }
2660
2661    if (mode==1) {
2662      if((af=fopen("initial_area.txt","a+"))== 0L)
2663         return (0);
2664      if((sh=fopen("scale_history.txt","a+"))== 0L)
2665         return (0);
2666      for (i=0;i<phases.no_phases;i++) {
2667        fprintf(sh, "%le ", phases.phasep[i].scale);
2668      }
2669      fprintf(sh, "\n");
2670      fclose(sh);
2671      for (i=0;i<phases.no_phases;i++) {
2672        fprintf(af, "%1.12lf ", area[i]);
2673      }
2674      //fseek(af, -1, SEEK_CUR);
2675      fprintf(af, "\n");
2676      fclose(af);
2677    }
2678    else if (mode==2) {
2679      if((af=fopen("initial_area.txt","r"))== 0L)
2680         return (0);
2681      if((sh=fopen("scale_history.txt","a+"))== 0L)
2682         return (0);
2683  //   fseek(af, -1, SEEK_END);
2684  //
2685  //   while (found==0) {
2686  //     c = fgetc(af);
2687  //     //printf("%d\n", c);
2688  //     if (c=='*') {
2689  //       found=1;
2690  //     }
2691  //     else
2692  //       fseek(af, -2, SEEK_CUR);
2693  //   }
2694
2695
2696       for (i=0;i<phases.no_phases;i++) {
2697        x = fscanf(af,"%lf",&org_area[i]);
2698        if(x==0) {
2699          cout << "ERROR: reading initial_area file" << endl;
2700          break;
2701        }
2702       }
2703  //   cout << "Initial Area: Phase1: " << org_area[0] << ", Phase2: " << org_area[1] << endl;
2704       fclose(af);
2705
2706       for (i=0;i<phases.no_phases;i++) {
2707        diff=100*area[i];
2708        fprintf(sh, "%lf ", diff);
2709  //      if (diff>0)
2710  //        phases.phasep[i].scale = phases.phasep[i].scale*fabs(diff);//, phases.phasep[i].elasticity);
2711  //      else if (diff<0)
2712  //        phases.phasep[i].scale = phases.phasep[i].scale*fabs(diff);//, phases.phasep[i].elasticity);
2713       }
2714
2715  //   for (i=0;i<phases.no_phases;i++) {
2716  //     fprintf(sh, "# %le %le #", phases.phasep[i].scale, area[i]);
2717  //   }
2718       fprintf(sh, "\n");
2719       fclose(sh);
2720
2721
2722
```

```
2723      }
2724
2725      return 1;
2726    }
2727
2728    int setupflynnies(void)
2729    {
2730
2731      int max, i, temp_int;
2732      double temp_double;
2733
2734      vector<int> ran;
2735
2736      max = ElleMaxFlynns();
2737      ran.clear();
2738      for (i=0;i<max;i++)
2739        if (ElleFlynnIsActive(i))
2740          ran.push_back(i);
2741      max=ran.size();
2742
2743      if ((grains = (Flynnies *)malloc(max*sizeof(Flynnies)))== 0)
2744        printf("ERROR: diffusearea: Malloc_Err: flynns\n");
2745
2746      for (i=0;i<max;i++) {
2747        grains[i].flynn=ran.at(i);
2748        ElleGetFlynnRealAttribute(grains[i].flynn, &temp_double, iFlynnPhase);
2749        temp_int = (int)temp_double;
2750        grains[i].phase=temp_int;
2751      }
2752      return max;
2753    }
2754
2755    clusters::clusters ( vector<int> vPushedFLynns, double dPushedArea )
2756    {
2757      //use swap pointers if possible put the pushed flynns in class storage.
2758    }
2759
2760    clusters::~clusters ()
2761    {
2762
2763    }
```

APPENDIX 6

CONTENTS OF THE DVD

CODE

This directory contains the source code of all developments that during my PhD.

- **split2** → is the new version of the splitting code
- **jr_gg_split** → is the combination of the growth and the split2 code that was used in the first project.
- **gbm_tou** → was the first, geometrical approach to two phase grain growth.
- **gbm_pp_old** → is the second approach to poly phase grain growth using the boundary nodes to keep areas constant.
- **gbm_pp** → the last, most recent approach to poly phase grain growth utilising Flynns to keep areas constant.
- **fft** → The last version of the fft integration in Elle including adjustments that were made to make poly phase grain growth and fft work in conjunction with each other.
- **jr_collection**, **jr-stats**, **ellefilecreator**, ***.py scripts** → these folders contain tools that I wrote to make the daily workflow easier. Description is given in appendix 4.

CONFERENCES + WORKSHOPS

This directory contains several directories related to conferences and workshops. These contain material that was used to prepare talks and posters for these conferences.

EXPERIMENTS + ANALYSIS

All data regarding the experiments can be found here. The analysis is also included here.

- **gg+split** → contains files from the first project. Read the explanation.txt files in the individual directories for more information on folder structure.
- **ice+air-bubbles** → contains files from the second project. Read the explanation.txt files in the individual directories for more information on the folder structure and experiment settings.
- **fft** → contains files that were used for the third publication and will also be used to complete the third project.

THESIS

The digital version of this thesis can be found here along with digital versions of the publications.