UNIVERSITÉ DE GENÈVE
Département d'informatique

FACULTÉ DES SCIENCES
Professeur P. Zanella
Docteur B. Chopard

A new numerical approach to snow transport and deposition by wind: a parallel lattice gas model

THÈSE

présentée à la Faculté des sciences de Université de Genève
pour obtenir le grade de Docteur ès sciences, mention informatique

par

Alexandre MASSELOT

de
Zürich

Thèse N° 3175

GENÈVE
2000

Avant-propos

This dissertation presents the main subject of my work under the supervision of Dr. Bastien Chopard in the Group for Parallel and Scientific Computing of the Computer Science Department (CUI) at the University of Geneva. The subject concerns original numerical simulation techniques of the generic problem of erosion, transport and deposition of snow by wind. The most important outcomes were published in scientific journals and regularly presented at conferences during the research period between 1995 and 1999. During this period, the approach was several times reinvented or reformulated as it is expected for this kind of research. One of the actual contribution of the present publication is the global and coherent presentation of all the aspects of the approach and how it proposes a new unified method to a problem which has often been treated with dedicated, case-dependent techniques. Moreover, our method is likely to be applied in a context very different from the framework it is issued from, and thus I also briefly presents some promising new perspectives.

I did not dedicate a specific chapter to a state-of-the-art presentation of other solid particles trasnport by fluid techniques. Neither do I resume basic notions of cellular automata and parallel computing to be found in numerous text books. I choose to concentrated my efforts on our new results and aspects of our approach without forgetting to give significant references for the reader who may go further ahead. We believe and we hope that this may promote an accurate use of our simulation technique and serve as a basis for future studies.

Besides this work, at least three other less extensive researches were undertaken during the same period by the author, and the resulting publications are referred in the text and given in appendix. Although the subjects were completely different, ranging from smoker/non-smoker populations problems to theoretical physics questions, the same intensive usage of parallel algorithms and parallel computing resources, of generalized cellular automata-like numerical models and of a fundamental, say a physicist's, approach to complexity are characterizing all the works.

I would like to thank Doctor Bastien Chopard for accepting me in his research group and giving the opportunity of doing a PhD at the University of Geneva in the best possible human, scientific and technical conditions. This work was mainly supported by the Fond National Suisse de la Recherche. A special thanks is also addressed to Dr. Dieter Issler, who helped launching the project and pointing out the challenges, to Olivier Guex and Charly Wuilloud, from the Service de Forêts et du Paysage (canton du Valais) for their propositions of real outdoor challenges and the interest the have shown in the current work.

And I should not to forget my PhD students collegues Danuta Sosnowska, Alexandre Dupuis, Marc “Hack” Martin, Pascal Luthi, Paul Albuquerque, Pierre-Antoine Queloz, Rodolphe Chatagny, Stefan Marconi, Thierry Zwissig and other members of the CUI for their inestimable and very efficient help and presence during these years, developping a very fruitful working atmosphere.

Alexandre Masselot Geneva, March 2000

Résumé

Introduction

Le transport de la neige par le vent, et surtout le dépôt de la neige sous l'action du vent est un problème qui revêt de multiples facettes. Du point de vue de l'observation, quel est le lien entre la formation de rides à l'échelle du centimètre et celle d'une corniche de plusieurs dizaine de mètres? Du point de vue phénoménologique le transport de particules de neige par le vent peut comporter simultanément trois modes de transport (reptation, saltation et suspension, cf. figure 1).

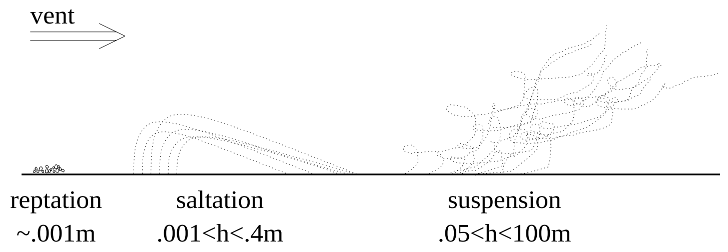


Figure 1: Traditionnellement, le tranport de particules de neige par le vent est partagé en trois modes: reptation au niveau du sol, trajectoires balistiques de saltation et suspension sur de plus larges distances [Castelle 1995].

Aborder un tel phénomène, dans le but de développer un outil numérique offrant une approche générale du problème, est dès lors un véritable défi.

Cependant, des enjeux liés à la vie courante sont une motivation supplémentaire pour aborder un problème déjà très riche au niveau théorique.

Par exemple, la structure d'un dépôt autour d'une crête alpine, avec le développement d'une corniche peut s'avérer un danger pour des habitations, des pistes de ski ou une route en aval de cette crête. Simuler ce dépôt, et surtout tenter de prédire l'influence de construction humaine pour combattre ce dépôt est un défi auxquels sont confrontés des experts de terrain.

Un très bref état de l'art

Certes, ce problème n'est pas nouveau, et de nombreux auteurs se sont penchés sur des aspects théoriques concernant la quantification du transport [Anderson *et al.* 1991], la simulation de dépôts par le biais d'expériences en soufflerie (autour de barrières [Iversen 1980], ou sur un col alpin [Castelle 1995]).

Depuis une quinzaine d'années, l'outil numérique est aussi apparu dans ce domaine. S'il s'est s'agit dans un premier temps d'aborder des cas précis par des méthodes *ad-hoc* (basées sur des statistiques ou avec un champ de vent préalablement fixé par exemples), des approches plus générales ont émergé: basées sur des solveurs de fluide classiques (souvent commerciaux, comme Flow 3D), il s'agit d'ajouter une phase solide. Confrontées à la réalité, ces approches ont donné de bons résultats, pour des simulations de dépôts autour de lotissement [Sundsbo 1997] ou sur une crête alpine [Gauer 1999] par exemples.

Cependant, ces approches traditionnelles se basent sur une modélisation numérique de systèmes d'équations décrivant le phénomène (*i.e.* d'un modèle théorique), et plus précisément sur des systèmes d'équations restreints à certains sous-phénomènes (typiquement, seuls un ou deux modes de transport des particules - cf. figure 1 - sont pris en compte).

Une approche originale

Plutôt que de considérer la description d'un phénomène (typiquement l'écoulement d'un fluide) par le biais de grandeurs macroscopiques (la pression, la vitesse locales), une autre approche est possible: il s'agit de représenter le média (ici le fluide) par un ensemble de particules discrètes se déplaçant sur un réseau régulier, évoluant avec des règles locales et synchrones.

L'avantage de cette méthode est qu'il est possible d'incorporer la dynamique de ce système directement dans ces règles locales. Si une telle représentation semble au premier abord trop naïve pour modéliser correctement un phénomène complexe, il a été montré à plusieurs reprises comment le comportement global du système ne dépend que peu des détails de la représentation, pour peu que les composantes fondamentales du phénomène aient été incorporées dans les règles d'évolutions [Hillis 1989].

Dans les cas d'écoulements de fluides, de telles méthodes sont développées depuis une quinzaine d'années. Elles ont atteint une certaine maturité et offre de très bons résultats pour des écoulements complexes, turbulent, avec des configurations de solides évoluant avec le temps, semi-poreuses ... Il s'agit des modèles de gaz sur réseaux.

De plus, cette approche, basée sur des interactions locales, sur un réseau régulier, se prête idéalement à une implémentation sur des machines parallèles. Plutôt que de ne se servir que d'une machine mono-processeur, le principe est de partager le travail sur un grand nombre de processeurs connectés efficacement (typiquement une trentaine de PC liés par un switch) afin de diminuer d'autant les temps de calcul.

Le cadre du présent travail

Dans cette thèse, nous avons donc opté pour une telle approche. Il nous a fallu reprendre les travaux existants dans la littérature pour le fluide et développer de nouvelles idées pour la phase solide.

Reprenant brièvement le travail de thèse, ce résumé se partage en plusieurs parties:

- description du modèle de fluide, ainsi que son utilisation dans des cas pratiques en trois dimensions,
- description du modèle pour les particules solides, et leurs interactions avec le fluide,
- résultats de simulations de dépôts de neige, à plusieurs échelles,
- présentation de la parallélisation du modèle, ainsi que des gains liés à cette parallélisation,

Le fluide

L'échelle mésoscopique

Traditionnellement, un fluide (mais cela s'applique à de nombreux autres phénomènes), peut être décrit à deux échelles:

- **microscopique:** quand on individualise chacune des molécules du fluide, avec sa position, sa propre vitesse, les collisions avec d'autres molécules semblables (des modèles numériques existent, dits de *dynamique moléculaire*),
- **macroscopique:** quand, à plus large échelle, on mesure la pression, la vitesse du fluide, dans une cellule de l'espace.

Du point de vue de la simulation numérique à large échelle, la première méthode est irréaliste au regard du trop grand nombre de particules à prendre en compte et de la complexité algorithmique d'un tel problème. La seconde approche est plus classique: en se basant sur les équations de *Navier-Stokes* qui décrivent l'écoulement d'un fluide, des schémas numériques plus ou moins complexes permettent de simuler un flux. Cependant, une telle approche se révèle rapidement complexe dans les cas d'écoulements turbulents: par dela une résolution d'équations différentielle, sur une grille hétérogène (avec une résolution plus fine aux endroits critiques), il faut souvent coupler un modèle de turbulence. Cette approche est néanmoins utilisée dans la grande majorité des problèmes concrets liés à la mécanique des fluides.

Une autre approche existe, se basant sur le fait que plusieurs niveaux de réalité existent en physique [Kadanoff 1986]. Un résultat important de la mécanique statistique est que le comportement à l'échelle macroscopique d'un phénomène ne dépend que peut des détails des interactions microscopiques. On peut donc utiliser cette proprieté pour construire un univers fictif, à une échelle **mésoscopique**, particulièrement bien adapté à la simulation numérique, où les ingrédients de base du phénomène sont pris en compte et donc le comportement macroscopique recouvert.

Depuis une quinzaine d'années, ces méthodes on été adaptées au cadre de la dynamique des fluides. Des premiers *automates cellulaires* discrets, bien des améliorations ont été imaginées pour arriver aux modèles de Boltzmann sur réseau, incorporant des techniques d'extrapolations sous-grille (*subgrid models*). Nous présentons ici un bref aperçu de cette évolution, par ailleurs très bien décrite par [Qian *et al.* 1996a, Chopard *et* Droz 1998].

Automates cellulaires

Plutôt que de présenter une introduction générale sur les automates cellulaires, nous allons nous contenter seulement des aspects intéressant le cas de certains modèles dédiés à la représentation de fluides.

Quelques définitions

Un atomate cellulaire est un système fictif dans lequel:

- l'espace est représenté par un réseau régulier Γ;
- chaque noeud $\mathbf{r}$ de ce réseau est lié à q voisins $\{\mathbf{r}+\mathbf{c}_i\}_{i=0,q-1}$;
- l'état de chaque cellule (ou noeud) $F(\mathbf{r},t)$, t est discret, appartient à un ensemble fini de valeurs; dans le cas présents, il s'agit généralement de $F(\mathbf{r},t)=\{F_i(\mathbf{r},t)\}_{i=0,q-1}\in[0,1]^q$, où $F_i(\mathbf{r},t)=0/1$ indique l'absence/présence d'une particule discrète sur le site $\mathbf{r}$, au temps t, avec vitesse $\mathbf{c}_i$;

Le fluide étant défini comme un ensemble de particules discrètes se déplaçant d'une manière synchrone sur un réseau régulier, on peut alors définir deux quantités locales: la densité

$$\rho(\mathbf{r},t)=\sum_{i=0}^{q_f-1}F_i(\mathbf{r},t) \qquad (0.1)$$

et la quantité de mouvement, *i.e.* le nombre de particules multiplié par la vitess sur la cellule

$$\mathbf{J}(\mathbf{r},t) = \sum_{i=0}^{q_f-1} F_i(\mathbf{r},t)\mathbf{c}_i = \rho(\mathbf{r},t)\mathbf{u}(\mathbf{r},t) \tag{0.2}$$

Règles d'évolution

L'évolution d'un automate cellulaire est synchrone, et peut s'écrire sous la forme

$$F_i(\mathbf{r}+\mathbf{c}_i,t+1) = F_i(\mathbf{r},t) + \Omega F(\mathbf{r}+\mathbf{c}_i,t+1) \tag{0.3}$$

où Ω est l'opérateur de collision.

Une règle de collision, dans le cas du modèle FHP sur un réseau bi-dimensionnel hexagonal [Frisch *et al.* 1986], est montrée dans la figure 2. Les règles fixées dans le cas de la modélisation d'un fluide doivent satisfaire deux critères:

- consevation de la masse (cf. équation 0.1,
- conservation de la quantité de mouvement (cf. équation 0.2.

De plus, certains autres critères doivent être respectés, comme des conditions d'isotropie du réseau (un réseau à quatre voisins ne serait pas correct), la conservation, en moyenne, de l'energie ...

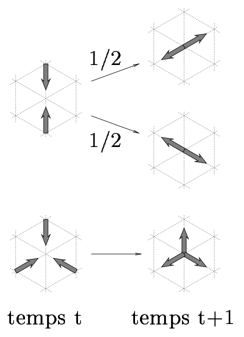


Figure 2: règle de collision FHP, sur un réseau hexagonal. Quand deux particules entrent sur le même site avec des vitesses oppposées, elles sont déviées avec une probabilité 1/2 de 60 ou -60 degrés. Quand trois particules entrent sur un site, avec une quantité de mouvement totale nulle, leur vitesses sont inversées. Toutes les autres configurations restent inchangées.

Implémentation

La réalisation d'un automate cellulaire se prête idéalement à l'implémentation sur un machine. Outre le fait que le domaine soit représenté par un réseau régulier, l'état par une nombre fixe de booléens, le confort vient aussi de la simplicité de l'algorithme qui, à chaque pas de temps, se décompose en deux étapes:

1. calcul de la nouvelle distribution,

2. propagation de l'information aux voisins.

De plus, l'incorporation de sites solides ne posent aucun problème: au lieu d'interagir normalement, les particules de fluides entrant sur un site solide voient leurs vitesses inversées (*no-slip* ou *bouncing back* conditions). Un évolution à plus large échelle d'un système est donnée dans la figure 3

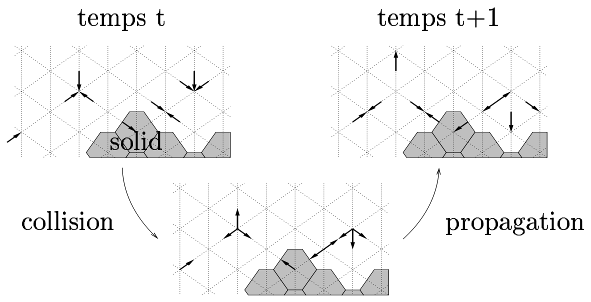


Figure 3: un pas d'évolution d'un modèle FHP, décomposé en deux étapes: a) collision b) propagation de l'information aux voisins.

Les modèles de Boltzmann sur réseaux

S'ils sont attirants, de par leur simplicité algorithmique, leur esthétique, les automates cellulaires pêchent par quelques défauts, principalement:

- un bruit statistique trop important, requérant des moyennes à large échelle,

- l'impossibilité de fixer finement des conditions aux bords.

Pour résumer très brièvement, deux étapes majeures marquent l'évolution des premiers automates cellulaires vers des modèles modernes, simulant des écoulements turbulents.

Le premier pas est de ne plus considèrer la présence/absence d'une particule $F_i(\mathbf{r}, t)$ comme une variable booléenne, mais plutôt comme une *probabilité de présence* $f_i(\mathbf{r}, t) \in \{0, 1\}$. Si les corrélations à n-corps sont négligées (cela peut-être le cas pour de nombreux problèmes), la règle de collision booléenne peut directement être traduite en une règle sur des variables réelles (les opérateurs *and*, *or* et *not* sont alors simplement remplacés par les opérations $\times$, $+$ et $(1 - .)$, la densité et la quantité de mouvement locales calculées de la même manière que dans les équations 0.1 et 0.2).

Cependant, cette méthode est intrinsèquement liée à la règle discrète sous-jacente et n'offre que peu de possibilité de modifier à volonté la viscosité du fluide (déduite de la règle de collision et la résolution du système).

Toujours plus haut (dans les nombres de Reynolds)

Aux méthodes de gaz sur réseaux se sont traditionnellement vu opposés les arguments: "la méthode est intuitivement esthétique, mais ne permet pas de simuler des écoulements de fluide intéressants (*i.e.* assez turbulents, avec des nombres de Reynolds assez grands)".

Un second pas majeur a été effectué quand, plutôt que de décrire une règle de collision déterminiée, la règle d'évolution consiste en une relaxation locale du système vers une *distribution à l'équilibre* $\{f_i^{eq}(\mathbf{r},t)\}_{i=0,q}$, avec un *temps de relaxation* τ:

$$f_i(\mathbf{r} + \mathbf{c}_i, t+1) = f_i(\mathbf{r},t) + \frac{1}{\tau}\left(f_i^{eq}(\rho(\mathbf{r},t), \mathbf{u}(\mathbf{r},t)) - f_i(\mathbf{r},t)\right) \tag{0.4}$$

où la distribution à l'équilibre ne dépend que la densité et la vitesse locales:

$$f_i^{eq} = \rho t_i \left[1 + \frac{\mathbf{c}_{i\alpha}\mathbf{u}_\alpha}{\mathbf{c_s}^2} + \frac{1}{2}\left(\frac{\mathbf{c}_{i\alpha}\mathbf{u}_\alpha}{\mathbf{c_s}^2}\right)^2 - \frac{\mathbf{u}_\alpha\mathbf{u}_\alpha}{2\mathbf{c_s}^2}\right] \tag{0.5}$$

oû t_i est un poids associé à la direction i du réseau ($t_i = 1\ \forall i$ dans le cas du réseau hexagonal), et $\mathbf{c_s}$ la *vitesse du son*, paramètre lié au modèle.

Un avantage de cette amélioration, du point de vue pratique est que le paramètre libre τ est directement lié à la viscosité ν du fluide:

$$\nu = \frac{\mathbf{c_s}^2}{2}(2\tau - 1) \tag{0.6}$$

Il est donc théoriquement possible, en faisant tendre τ vers 0.5, d'obtenir une viscosité arbitrairement petite (donc un nombre de Reynolds arbitrairement grand).

Cependant, des problèmes de stabilité numérique apparaissent si τ est trop petit. La solution, pour contourner ce problème a été d'adapter à la méthode des gaz sur réseau la technique LES (*Large Eddy Simulation*), *i.e.* d'extrapoler l'influence de ce qui se passe à des échelles plus petites que celles du réseau Γ. Il s'agit des modèles de sous-grilles (*subgrid model*), comme celui de Smagorinski (avec une constante C_{smago}), stables, avec lesquels des nombres de Reynolds de l'ordre de 10^6 peuvent être atteints [Hou 1995].

Il est intéressant de noter que, dans ce cas, le code d'un tel programme reste intrinsèquement très simple, et qu'il suffit de quelques dizaines de lignes pour écrire un modèle de fluide en trois dimensions, rapide, avec des frontières solides qui peuvent naturellement évoluer.

Quelques résultats

De nombreuses expériences ont été menées, notament pour explorer certains aspects pratiques du modèle tri-dimensionel. Nous en présentons ici deux:

- un écoulement derrière un cylindre (figure 4),
- deux écoulements dans un tuyau, pour montrer l'efficacité du modèle de sous-grille (l'écoulement moyen est le même, mais le modèle sous-grille développe une beaucoup plus grande activité turbulente), figure 5.

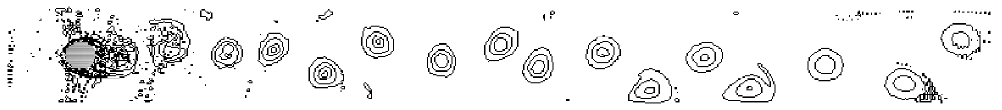


Figure 4: écoulement derrière un cylindre, avec le développement d'une allée de Von Karman.

Les particules solides

La contribution principale de ce travail réside dans l'incorporation d'une phase solide, dans le but de simuler l'érosion, le transport et le dépôt de particules de neige par le vent.

L'utilisation d'un modèle de fluide sur réseau, par sa simplicité, son efficacité et sa souplesse ne constitue en fait qu'un préliminaire.

Cependant, la philosophie des modèles sur réseaux, avec la création d'un monde fictif et l'implémentation directe de règles intuitives reste de mise pour cette partie. En effet, plutôt que de tenter d'adapter des schémas théoriques classiques (quantifiant, le taux d'érosion, la diffusion des particules pendant le transport ...), nous avons directement imaginé des règles régissant le comportement individuel des particules.

Représentation

Sur le même réseau Γ que pour le fluide, les particules solides, sur le site $\mathbf{r}$, au temps t, voyageant dans la direction $\mathbf{c}_i$ sont représentées par $p_i(\mathbf{r}, t) \in \mathbb{N}$, et $p_{rest}(\mathbf{r}, t)$ contient le nombre de particules au repos sur le site.

Ces particules solides voient leur mouvement dirigé par la vitesse locale du vent et la gravité. Trois actions doivent donc être implémentées:

- le transport éolien,
- le dépôt,
- l'érosion.

Le transport éolien

Les particules sur le site (non-solide) $\mathbf{r}$ subissent la vitesse locale du fluide $\mathbf{u}(\mathbf{r}, t) + \mathbf{u}_{fall}$, où $\mathbf{u}_{fall}$ est la vitesse de chute. Cependant, si les particules suivent exactement cette vitesse, il est très peu probable que la destination soit un site du réseau. Or les particules doivent rester sur le réseau.

Nous avons donc imaginé un algorithme probabiliste de transport éolien des particules qui satisfait en moyenne un transport exact. Cet algorithme est expliqué pour les cas à deux dimension dans la figure 6.

Il est intéressant de noter que si avec cette méthode aucune différence n'est faite entre les différents modes de transport (reptation, saltation et suspension), des expériences montrent comment ils sont naturellement générés.

Le dépôt

Quand une particule veut se déplacer vers un site solide (*i.e.* dans direction $\mathbf{c}_i$ depuis le site $\mathbf{r}$ si le site $\mathbf{r} + \mathbf{c}_i$ est solide), elle doit se déposer. Dans notre nomenclature, nous dirons que cette particules est gelée.

Quand un nombre θ_{frz} de particules gelées est atteint sur une cellule, cette cellule est solidifiée.

Cependant, l'état solide n'est pas définitif, puisque des particules peuvent être érodées.

L'érosion

Les mécanismes liés à l'érosion sont nombreux, et souvent controversés. Plusieurs techniques ont été testées, mais une idée simplifiée a été gardée. Il s'agit d'éjecter les particules de la couche supérieure du dépôt avec une probabilité donnée ζ_p (cette probabilité peut être multipliée par la vorticité locale du fluide pour prendre en compte les micro-tourbillons). Une fois éjectées, les particules sont soumises au transport éolien décrit ci-dessus, et peuvent donc soit être tranportées, soit retomber à leur place originale (si le vent local est faible, il n'y a donc pas d'érosion effective).

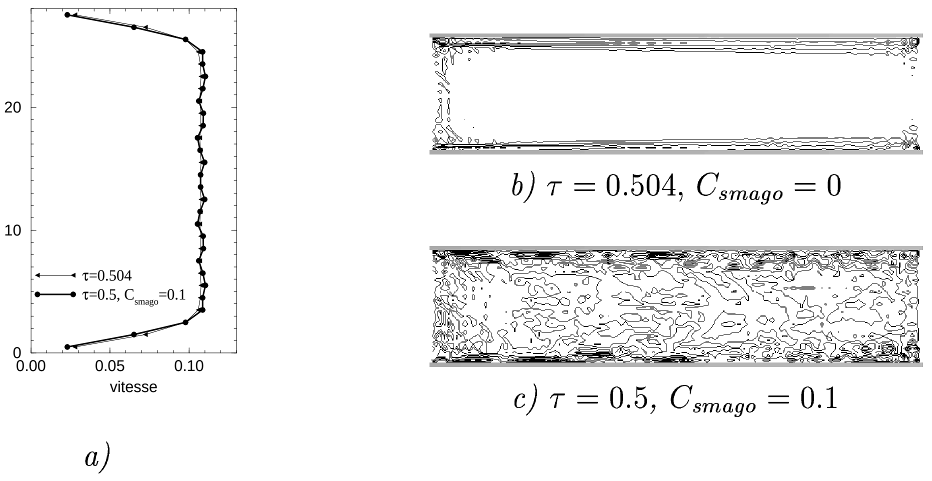
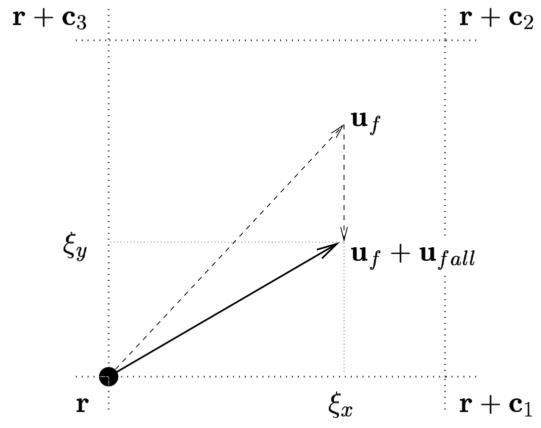


Figure 5: écoulement dans un tuyau (coupe d'une expérience en trois dimensions), avec ou sans l'utilisation du modèle de sous-grille de Smagorinski. Le graphique *a)* montre que les deux profils de vitesses moyennés sont identiques. Cependant, l'affichage des champs de vorticité instantanés montre comment le modèle de Smagorinski fait apparaître un développement des tourbillons (figure *c)*) quand le modèle classique ne calcule qu'un champ moyenné (figure *b)*).



$\mathbf{r} + \mathbf{c}_3$
$\mathbf{r} + \mathbf{c}_2$
$\mathbf{u}_f$
ξ_y
$\mathbf{u}_f + \mathbf{u}_{fall}$
$\mathbf{r}$
ξ_x
$\mathbf{r} + \mathbf{c}_1$

Figure 6: la position exacte des particules du site $\mathbf{r}$ au temps $t+1$ est $\mathbf{r}+\mathbf{u}(\mathbf{r},t)+\mathbf{u}_{fall}$, qui n'est généralement pas un site du réseau. Pour obtenir une déplacement qui soit, *en moyenne*, correct, les particules sont distribuées aléatoirement entre les quatre voisins les plus proches de la destination finale. Dans le cas proposé, le transport dans la direction x s'effectue avec probabilité ξ_x, et en y avec ξ_y. En conclusion une particule se déplace vers $\mathbf{r}+\mathbf{c}_2$ avec probabilité $\xi_x\xi_y$ (déplacement en x et en y), vers $\mathbf{r}+c_{1,3}$ avec $\omega_{1,3} = \xi_{x,y}(1-\xi_{y,x})$ (déplacement en x et pas en y ou vice-versa) et reste sur place avec $(1-\xi_x)(1-\xi_y)$ (déplacement ni en x, ni en y).

Résultats

Si le modèle de fluide peut être justifié théoriquement (il satisfait les équations de Navier-Stokes), il n'en est pas de même pour celui concernant la phase solide. Pour montrer la validité de la méthode, il faut donc montrer des résultats satisfaisants.

Contrairement à une simulation classique, en créant un monde fictif, la relation entre l'échelle modélisée et la réalité n'est pas évidente (quelle est la taille réelle d'une cellule? quel est le pas de temps?).

Puisque le modèle n'est pas directement lié à une échelle d'espace, il n'existe pas, *a priori*, une échelle privilégiée. Partant de cette observation, nous avons mené de nombreuses expériences, dont nous présentons quatre aperçus ici:

- formation de rides à partir d'un dépôt plat,
- distributions des longueurs de saut de particules, comparées à des expériences en extérieur pour plusieurs vitesses de vent,
- dépôts autour de routes à flanc de côteau, et l'influence d'ouvrages,
- dépôts au niveau de crêtes alpines, et la prédiction de l'influence de toits-buses.

Rides

En partantd'un dépôt complètement plat, on observe dans la réalité la formation de rides (micro-dunes, ou *ripples*). Confronté à cette situation, notre modèle en développe, et la reptation de ces formations peut être observée sur la figure 7.

Distributions de longueurs de saut de particules

Dans le mécanisme de transport des particules fictives, aucun ingrédient ne précise s'il s'agit de reptation, de saltation ou de suspension. Dans la réalité, une expérience permet de quantifier la longueur des sauts [Kobayashi 1972]: il s'agit de disposer des réceptacles métalliques de taille fixe (L cm) dans le sens du vent: pour arriver dans la $n^{\text{ème}}$ boite, une particule de neige a du faire un saut d'au moins $n \times L$ cm. Après un certain temps, il suffit de mesurer combien de particules sont arrivées respectivement dans les boites pour en déduire une distribution de la longueur des sauts.

Une telle expérience est facile à reproduire avec notre modèle, et les résultats sont comparés à la réalité avec une grande satisfaction, pour deux vitesses de vent différentes, dans la figure 8.

Routes à flanc de côteau

Pour deux situations tirées de [Castelle *et al.* 1992], nous avons confronté notre modèle aux observations sur le terrain, ainsi qu'a des aménagements imaginés (pas forcément les meilleurs, mais utiles pour se rendre compte de la valeur des autres prévisions):

1. route à flanc de coteau bordée par un talus, quand le vent descend la pente; le talus peut être aplani, ou différents toits buses plus ou moins intelligemment proposés (figure 9);

2. même situation, mais le vent remonte la pente (figure 10).

Si les résultats sont satisfaisants avec le premier cas, il ne sont parfaits dans le second, dans la mesure où le modèle ne "saisit" pas bien la finesse du premier dépôt. Ce problème aurait pu être réglé en changeant certains paramètres, cependant, associer chaque expérience avec un nouveau réglage empirique est en contradiction avec une vue unifiée du problème.

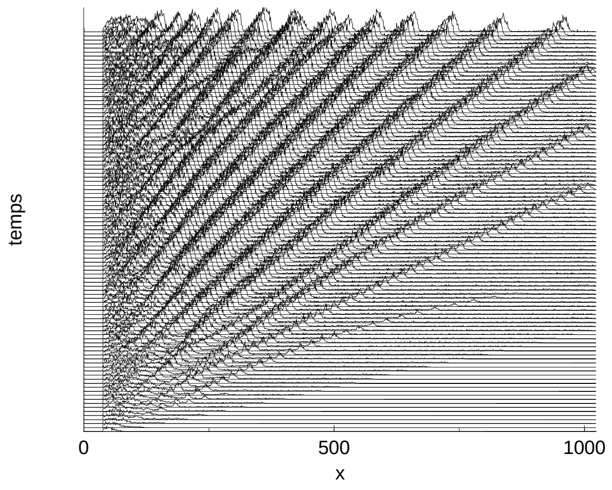


Figure 7: formation de rides. L'évolution du dépôt dans le temps monter la lente reptation du profil.

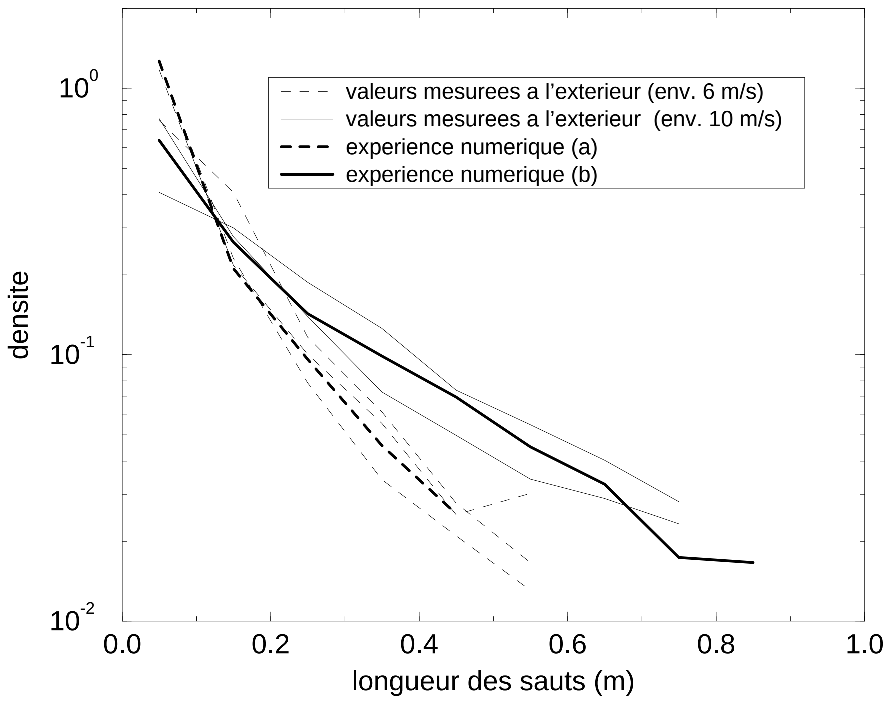
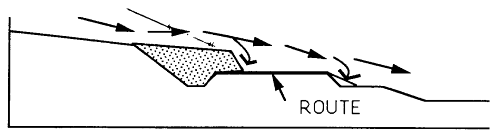
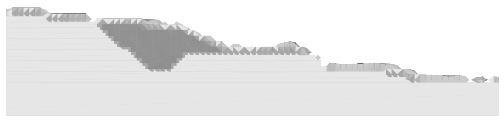
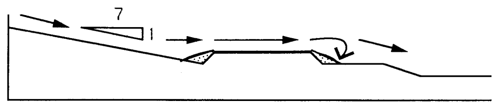


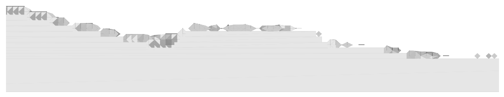
Figure 8: Distributions des longueurs de saut de particules. Les lignes pointillées représentent des expériences menées avec un vent deux fois plus fort que dans le cas des lignes continues. Les lignes grasses représentent les résultats numériques, qui peuvent être ici comparés à ceux du terrain [Kobayashi 1972].



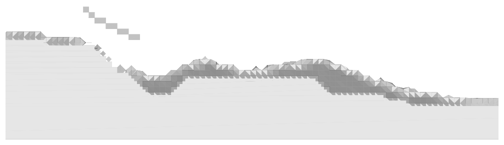


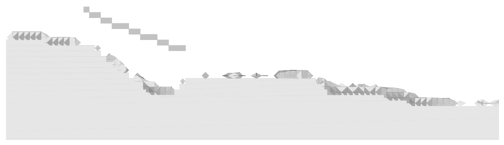
Dans les observations, comme avec le modèle, une congère bloque la moitié de la route. Sur les résultats du modèle, le solide est gris clair, la densité de particules solidifiées varie de gris foncé pour le maximum, à gris clair pour le minimum.





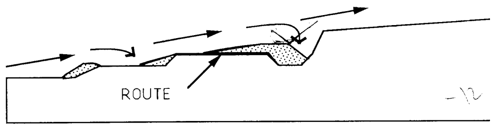
Aplanir le talus évite la congère, comme le montre aussi les simulations, cependant, d'un point de vue pratique, cette solution peut s'avérer difficile. Le dépôt clair sur la route signifie seulement une concentration faible à cet emplacement.

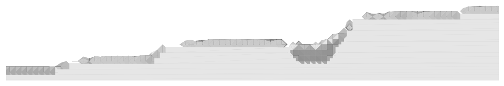




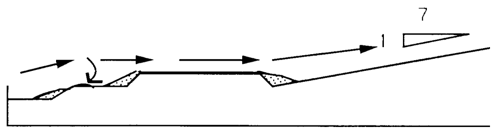
Deux toits buses sont proposés. Dans le premier cas, le fossé est déblayé, mais une congère apparaît au milieu de la route; dans le second cas, un plus long toit évite le problème.

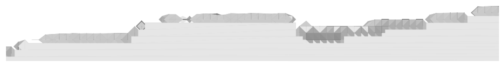
Figure 9: Divers aménagements autour d'une route à flanc de coteau, quand le vent descend la pente.



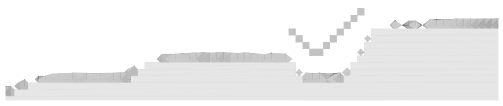


La congère, observée dans la réalité, est seulement partiellement retrouvée par le modèle.





La situation peut être améliorée en aplanissant le talus.



un ouvrage peu réaliste est proposé. Cependant, s'il dégage le fossé, il crée un dépôt sensiblement plus haut que normal au milieu de la route.

Figure 10: Une route à flanc de coteau bordée par un talus, quand le vent remonte la pente.

Crêtes alpines

A plus large échelle, nous avons confronté notre modèle à des simulations sur des crêtes alpines.

Avant de passer au problème dit de la Marlennaz, nous avons testé le modèle sur des crêtes pour lesquelles des résultats existent, la Schwarzhorngrat [Föhn *et* Meister 1983]. En effet, seuls des résultats sur un cas connu peuvent lever certains doutes quant à la fiabilité du modèle.

Schwarzhorngrat

D'autres cas existent dans la littérature, cependant, nous avons choisi celui-là car il offre trois avantages:

1. le vent dominant est perpendiculaire à la crête;

2. le dépôt a crû de manière monotone pendant les deux mois des mesures; en effet, il semble difficile de prévoir un dépôt dans le cas de conditions trop changeantes;

3. des mesures ont été effectuées le long de la crête sur trois profils différents; cette situation offre une banc d'essai d'un grande valeur.

Les résultats du modèle numérique sur les trois crêtes sont présentés vis-à-vis des mesures sur le terrain dans la figure 11. La comparaison est très bonne.

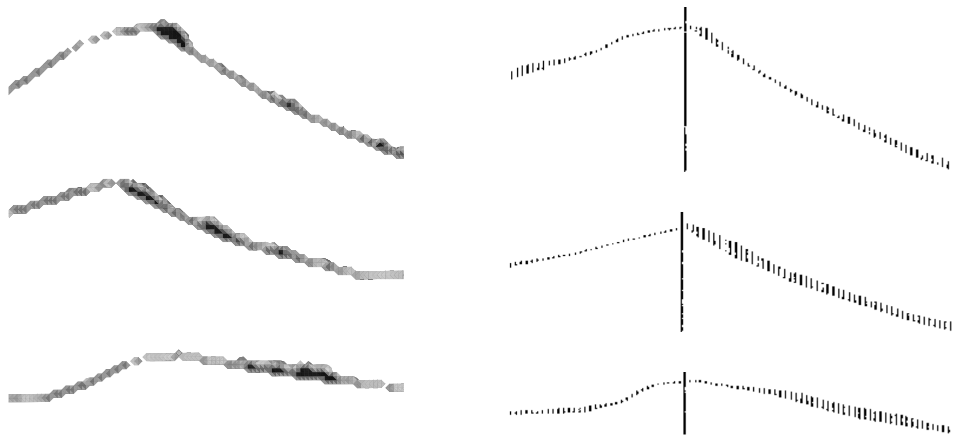


Figure 11: les comparaisons entre résultats numériques (à gauche) et observations sur le terrain de la Schwarzhorngrat [Föhn *et* Meister 1983] sont satisfaisants.

La Marlennaz

La situation offre un plus grand défi car, d'une part il n'existe pas de mesures de dépôts comparables au cas précédent, et d'autre part il s'agit de modéliser l'effet de toits buses dont le but est d'éviter la croissance d'une corniche.

Des simulations ont été produites sur les profils nommés respectivement 1 et 3.

Marlennaz - profil 1

Dans ce cas, des simulations ont été menées avec et sans toit buse: plusieurs étapes de la croissance du dépôt sont montrés dans la figure 12.

Marlennaz - profil 3

Ce profil offre deux difficultés:

1. sa configuration, avec une petite crête au vent dont l'influence est dure à estimer *a priori*, et plusieurs faibles ruptures de pente dans la pente principale, en font un profil complexe;

2. aucun ouvrage n'a encore été construit, donc la question de la position, du type d'ouvrage, reste complètement ouverte.

Des simulations de dépôts moyennés sont présentés dans la figure 13.

Conclusions sur les dépôts

Ces différents résultats, à travers une large gamme d'échelle spatiales, montre comment notre modèle, naïf au premier abord, a pu capturer des composantes essentielles d'une phénomène naturel aussi complexe que le transport de neige par le vent. Ils viennent justifier certains choix techniques du modèle numérique.

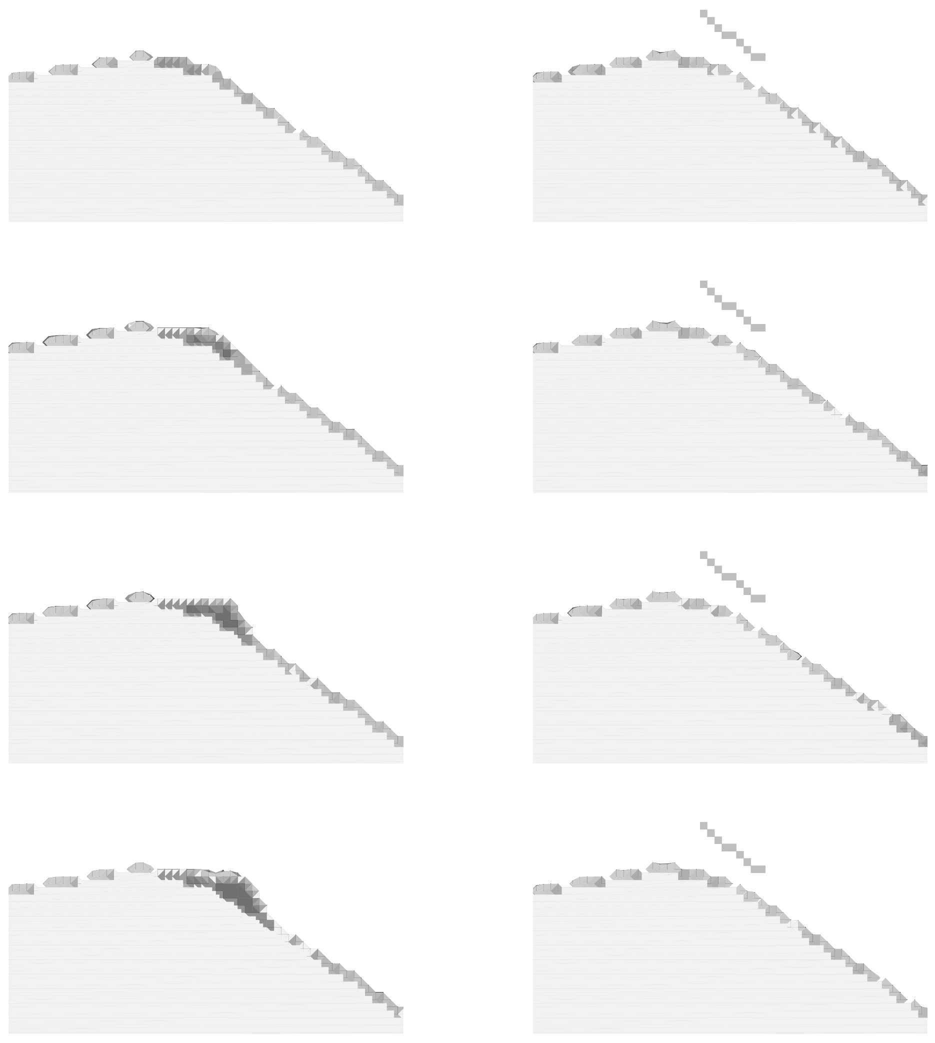


Figure 12: *La Marlennaz - profil 1.* Différentes étapes de la croissance du dépôt, avec ou sans toit buse. L'effet recherché de ce dernier (limiter la corniche) est clairement observé. D'autres positions pour le toit buses, moins efficaces, ont aussi été testées.

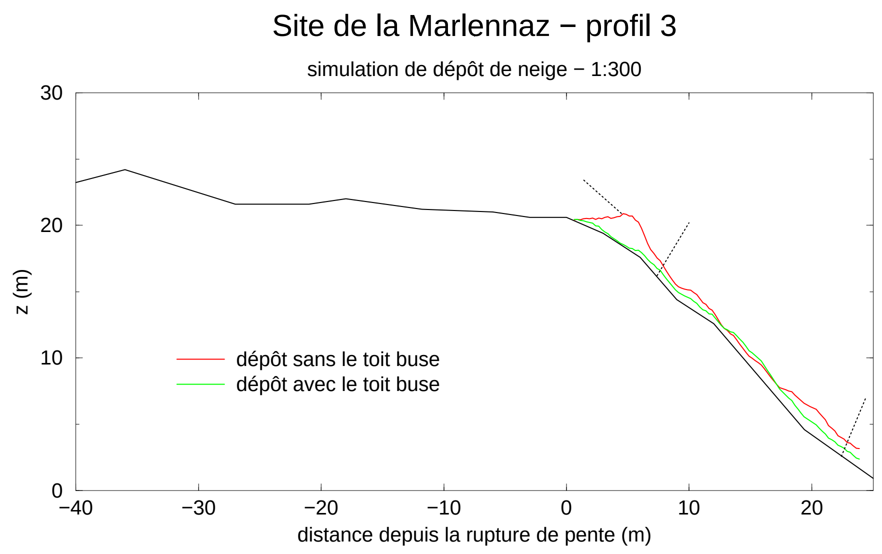


Figure 13: *La Marlennaz - profil 3.* Dépôts moyennés. La corniche est éliminée par le toit buse, mais un dépôt légèrement supérieur à la normale semble apparaître vers $x = 15$ mètres.

Quelques aspects informatiques

Fortran 90: un langage adapté au problème

Depuis l'avènement des languages orientés objet, le *Fortran* est généralement méprisé par les informaticiens. Victime de sa célébrité sous la forme de *Fortran 77*, lui sont reprochées toute une série de pêchés mortels: absence d'allocation dynamique, de types structurés, une indentation fixe ...

Cependant, comme les autres languages, Fortran a évolué pour atteindre l'état de *Fortran 90* (et avec quelques modifications mineures, *Fortran 95*).

Le but de ses créateurs n'a pas été d'élaborer un langage universel, titre auquel veulent prétendre plus sérieusement Java ou C++. Leurs priorités ont été les suivantes:

- la simplicité d'utilisation,
- une bonne gestion des problèmes liés à l'algèbre linéaire et aux problèmes mathématiques en général,
- rapidité d'exécution,
- fiabilité.

Le résultat est un langage modulaire (un module est un ensemble de variables et de procédures que partagent toutes les unités du programme qui *utilisent* ce module) qui remplit bien son cahier des charges.

Si la philosophie modulaire est aujourd'hui souvent considérée comme désuète, elle se révèle suffisante dans un cas comme le programme présenté dans cette thèse, pour structurer des données (un module pour le fluide, un pour les particules solides etc.) qui sont généralement peu complexes (en effet, un champs distribué sur un réseau régulier est stocké sous la forme d'un simple tableau multi-dimensionel).

La parallélisation du code

Pour exécuter plus rapidement un programme, la course au CPU le plus rapide est rapidement bornée (et très chère, accessoirement). Une solution est de partager le travail sur plusieurs CPU, pour diviser d'autant les temps d'exécution.

Les modèles comme ceux des gaz sur réseaux se parallélisent idéalement: il suffit de distribuer le domaine (un réseau régulier, donc un tableau) sur un ensemble de processeurs (cf. figure 14). Les comunications sont locales et régulières, et peuvent être implémentées avec des routines d'envoi de messages (*Message Passing Interface*).

Des mesures de scalabilité montrent, dans la figure 15, comment l'utilisation de n machines divise par n le temps d'exécution, dans le cas de l'utilisation d'un cluster de Pentium II à 500Mhz.

L'utilisation d'un language de programmation performant et adapté à ce problème (Fortran 90) et d'une librairie de communication standard (MPI) a permis de conduire des expériences sur plusieurs autres machines, notament une IBM SP2 à 14 processeurs RS/6000 et un cluster de 8 Sun Ultra 5.

L'utilisation de machines parallèles, avec des temps de réponse très court (la plus longue des expériences montrées dans ce résumé prend moins d'une quinzaine de minutes) permet un usage réellement interactif de l'outil numérique, et ainsi une exploration confortable des cas étudiés.

Cette interactivité modifie profondément les habitudes des numrériciens, traditionnellement confrontés à de fastidieux temps de calculs.

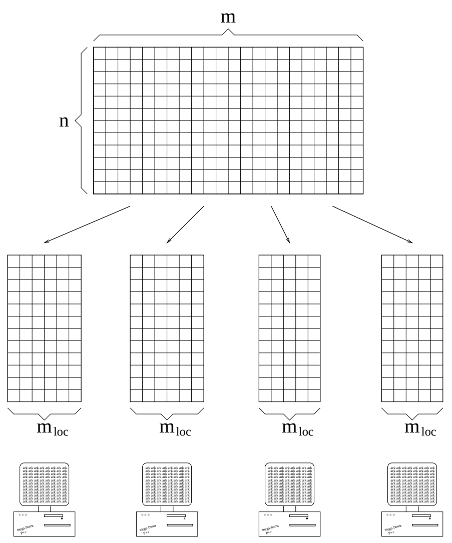


Figure 14: distribution d'un tableau de 22 colonnes sur 4 processeurs. Dans ce cas, deux processeurs gèrent 6 colonnes, et les deux autres 5 chacun.

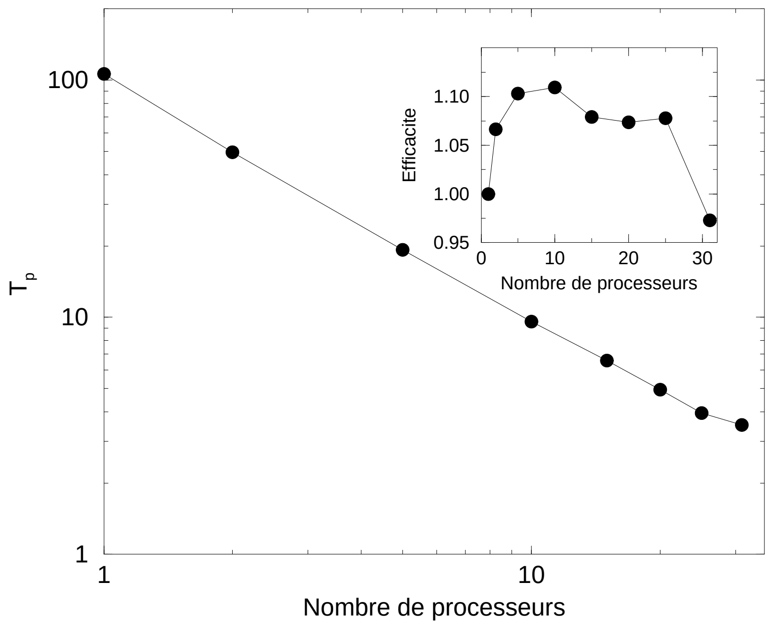


Figure 15: Scalabilité des performances sur un cluster de Pentium III. Le temps parallèle est indiqué en fonction du nombre de processeurs. Le sous-graphe montre l'efficacité de la parallélisation (une efficacité de 1 est idéale). Une efficacité supérieure à 1, comme dans le cas présent montre comment la parallélisation est superscalaire: en effet, le partage du problème en petites sous-tâches rend l'utilisation du cache plus performante.

Contents

List of Figures

Chapter 1

Introduction

In Alpine and Nordic countries, wind and snow have always been a major concern, at least in inhabited areas: obstacle to roads or railway lines which can be closed by snowdrifts more perniciously than by snow falls; threat in mountains, where wind slabs or cornices can trigger avalanches devastating inhabited places or closing roads.

To diminish these threats, men have built works, such as: fences, to store the snow deposit windward a road; wooden tunnels to protect railway lines (Bernina pass in Switzerland, Oslo-Bergen line in Norway); modifications of the landscape around Alpine roads; static paravalanches (one century old dry-stone walls above Davos or more recently iron net ones) and more innovative works, slanted screens or wind veering (at la Marlennaz near Verbier) or auto-orienting fences (at the Simplon pass).

These contructions can be organized in two categories:

- passive ones, not preventing the deposit, but protecting statically the target (wind tunnels, paravalanches);
- actives ones, changing the wind flow pattern, thus dynamically modifying the deposit shapes (*e.g.* fences, slanted screens, tree planting, landscape modifications).

As snow deposit patterns can be observed during previous winters, the main problem with passive works is not their location but the very high load of the snow and their building in hazardous areas (steep Alpine slopes); their conception is therefore a challenge for civil engineers and field workers.

Meanwhile, the load forces on active works are much lighter (high winds are the main stress). However, the choice of the most optimal type and work location, depending on the landscape shape is an non-trivial problem (different solutions to get rid of a snowdrift on a road are displayed in figure 1.1).
In standard situations (such as a flat plain with the prevailing wind blowing perpendicularly to a road), this choice can be based on statistical outdoor observations [Tabler 1980a, Tabler 1991]. However, when the case is not similar to

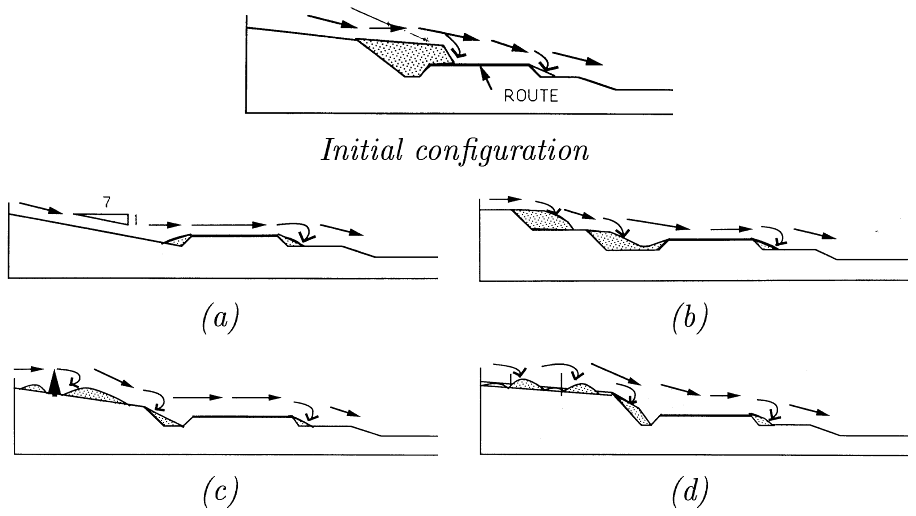


Figure 1.1: in the upper figure, a hill side road get obstructed by a snow drift when the prevailing wind blows downwards (from left to right) [Castelle *et al.* 1992]. Several actions can be undertaken to avoid this situation: *(a)* lowering the windwards slope; *(b)* building terraces; *(c)* planting trees and lowering the slope; *(d)* building snow fences. This case is more thoroughly studied in section 3.2.

litterature ones, the problem becomes more tricky: forecasting the influence of an obstacle on the wind pattern, thus on the deposit can be as chancy as forecasting next winter weather looking at onions peels. Based on his experience, a field expert has to imagined a solution, build it, wait several winters to test it, modify some parameters (location, works type or configuration), test it again ...

Since Johnson in 1852 (see figure 1.2), studying snow, and more precisely snow transport by wind and deposition, has therefore inspired many reseachers. A quick overview on the domain is summarized in the following section and an original method to model complex systems is introduced in the next ones.

1.1 Snow transport modelling: a state of the art

Presenting a global state of the art of snow transport and deposition studies is not the purpose of this thesis, more focused on a numerical model. However, an overview of this field is necessary to situate this work in its context (a more extensive panorama can be found in [Castelle 1995]). We will therefore introduce some basic concepts, and different approaches followed by other researchers.

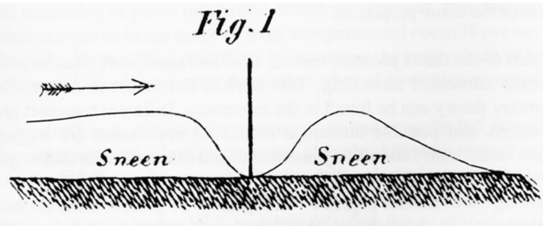


Fig. 1. Her viſes Snefogets Virkning over flad Mark, hvorpaa findes et fritſtaaende Plankeværk af almindeligt Slags, 10 Fod høit, med omtrent een Tommes indbyrdes Afſtand imellem ethvert lodretſtaaende Bord.

Figure 1.2: "the influence of snow drift over flat ground around an ordinary ten-foot fence, with one inch space between the vertical boards" (Johnson 1852) cited in [Sundsbo 1997]

1.1.1 Basic phenomenon

It is generally accpeted that snow is transported by wind following three different modes, in the very same manner as sand, displayed in figure 1.3 [Mellor 1965, Kobayashi 1972]:

- **creeping**: particles are rolling on the ground;
- **saltation**: they take off almost vertically, and fly along a parabolic trajectory; they can be ejected by strong enough eddy or by the shock of landing particles;
- **suspension**: during the saltation phase, particle can be taken in larger scale eddies and fly for longer distance.

Although the same components are present (turbulent wind flow and solid particles), these three modes are treated separately by all the authors. Each process is identified to a governing equation [Anderson *et al.* 1991] and prediction

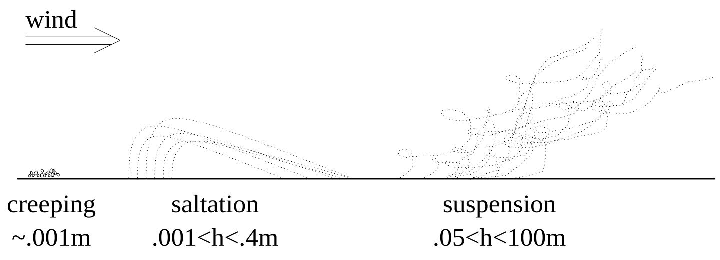


Figure 1.3: snow transport by wind is split into three modes: creeping of particles at ground level, ballistic saltation trajectories and suspension of particles over long distances [Castelle 1995].

models usually take into account only the main modes involved in the studied phenomenon.

This approach has been proven successful, for example, the erosion of a wind-facing step [Castelle 1995] or a numerical model for suspension transport [Martinez 1996]. However, beside these rather theoritical questions, many efforts (often by the same persons) have been achieved to answer field problems, such as the deposit around a fence, a building, an alpine crest etc. with the same kind of methods.

1.1.2 Modelling outdoor situations

Statistic or fully-dedicated models

Outdoor observations are recorded to build a statistics data base: this approach have been proposed by [Tabler 1980a] for deposits around fences. The prediction for a new case is therefore based on comparison with the nearest records. However, such a model can only be applied to a limited range of situations covered by the data base, and results can hardly be extrapolated.

Other forecasting tools, based on semi-empirical models have been developed:

- the deposit over an alpine governed by the addition of two transport modes: the potential (stationary) flow and a plume model for the turbulent non-steady effects [Föhn *et* Meister 1983];
- the situation in the canadian prairie decomposed in two processes: saltation and suspension, based on semi-empirical observations (*Prairie Blowing Snow Model* PBSM) [Pomeroy 1988].

However, this models are dedicated to particular cases, and cannot be directly used on more general problems. Another direction to explore was therefore the indoor wind tunnels.

Indoor wind tunnel

Perhaps the most natural approach, when predicting snow or sand deposits around structures, was to use indoor wind tunnels. However, some similarity criterion must be observed to recover the outdoor situation:

- Iversen have defined a set of criterias to model snow deposit around fences [Iversen 1980];
- the same ideas have been adapted to model snow erosion/deposition at an alpine pass [Castelle 1995].

However, since the advent of powerful computers, numerical wind tunnels are not out of reach anymore.

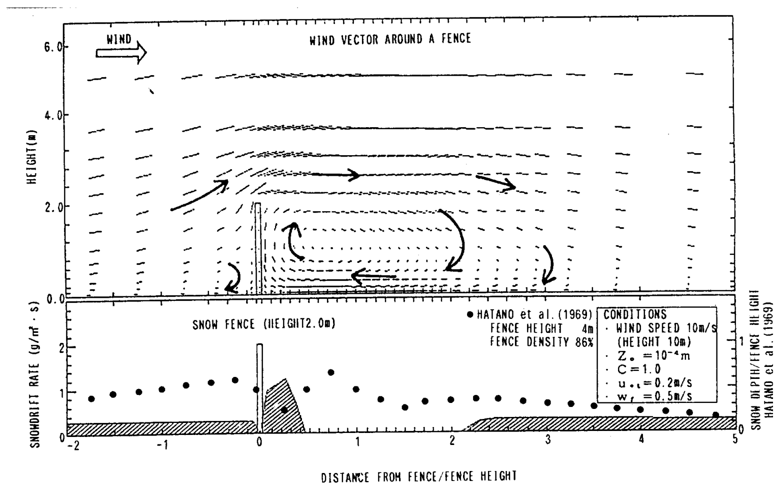


Figure 1.4: ouput from [Uematsu *et al.* 1991] around a fence, where wind flow is plot in the upper part; snow drift rate and snow depth are confronted to outdoor results in the lower part of the figure.

Classical numerical models

Very early, computers have been used to model fluid flows. It has therefore been natural to incorporate a solid phase to the fluid one and simulate the deposits.

One of the earliest three-dimensional models have been proposed by [Uematsu *et al.* 1991], and a result is shown in figure 1.4. The static wind flow, and the lack of an implementation of a particle saturation process cause some differences with reality. However, this model was applied to true 3D configurations, such as the deposit around a cube.

More recent approaches have used commercial fluid solvers, such as *Flow-3D*, and later added a solid particle phase. [Sundsbo 1997] focuses on snow deposit around man made structures, such as fences and overall 3D complex buildings and has proposed very accurate results. [Gauer 1999], with the same kind of physically-based approach, addresses alpine terrain questions, comparing fully 3D numerical simulations around large crest areas with field measurements.

As in many other fields, classical numerical models are directly based on theoritical ones. This approach can be broken down in three layers:

1. based on observations, the main components or of the phenomenon (at least the supposed most relevant ones) are pointed out;

2. following argument and deduction, these extracted variables are mixed into a set of governing equations;

3. to be numerically handled (for an extrapolation purpose for instance), these equations must be transformed into a computable form.

At each of these levels, simplifications are assumed, *a priori* decisions are taken. The more complicated the phenomenon is, the less a consensus can be reached by different approaches.

This chain process is of course not unique to the problem adressed in this thesis. However, transport of solid particles by a fluid is a very good example of a *complex system*. Therefore, the approach introduced in the following section, radically different from the classical numerical models and aimed to served a very wide range of purposes, suits perfectly our problem.

1.2 Modelling complex systems: lattice models

Modelling a complex system at the level of the elementary behavior of each of its components would soon become too complicated and out of reach of our knowledge and our computing resources. Indeed, looking closer at some examples quicly show their complexity:

- **chemical reaction:** each molecules of the species involved in the process has it own position, trajectory, solvent movement, light, pH, reactive competition, presence or absence or antagonists, ions, enzymes ...
- **prey-predator system:** each individual has a social position, physical abilities (therefore hunting efficiency, resistance ...), a feeding territory, knowledge, genetic capital; it can die, give birth, be a parasite vector ...
- **snow transport by wind:** humidity, shape of the flakes (and its modifications during time), evolution of the deposited snow characteritics (melting or freezing flakes), influence of landing particles on the bed, influence of the saltating density on the high frequencies of the fluid flow, particle toppling or cohesion ...

However, an alternative to the classical three layers thought process presented at the end of the previous section exists. We propose in the following sub-section a short introduction to *lattice models*, and we will briefly focus on the case of a chemical reaction. In section 2.1, we will go deeper into lattice models dedicated to fluids.

1.2.1 Building a fictitious world

This approach relies on the fact that several levels of reality exist in physics [Kadanoff 1986]. On one hand there is the macroscopic level, where phenomena are expressed in terms of rather abstract mathematical objects such as the differential equations. On the other hand, there is the microscopic level of description where the interactions between the basic constituents are considered.

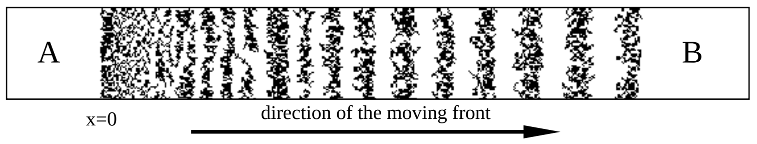


Figure 1.5: A particles diffuse (from left) in a solvent containing B particles; they precipitate into a new specy C, plotted in black in the figure. Agreeing with laboratory exepriments, a cellular automata models shows how C is produced in bands of growing width, the so-called *Liesegang bands*.

An important results of statistical mechanics is that the macroscopic level of description depends very little on the details of the microscopic interactions. One can use this property to build afictitious universe in which the microscopic interactions are particularly simple to simulate on a computer and whose macroscopic behavior is just that of the real system [Hillis 1989].

This approach has been applied to a very wide range of problems, *reaction-diffusion systems* briefly introduced in the following subsection [Cornell *et al.* 1991].

1.2.2 Reaction diffusion systems ...

Reaction-diffusion systems consist in the coexistence of diffusing chemical species, and reactions according to some condition, as for examples:

- chemical species A and B diffuse in a solvent, they can react and create a new species C, which can precipitate in turn;
- a species A diffuses and precipitates (into B) when it encounters a steady B species.

Although they are well understood from a chemical point of view (*i.e.* equations governing the process are clearly established), reaction-diffusion systems have been hard problems for computational models as they may produce complex behaviors: Liesegang bands for the first above example (cf. figure 1.5, dentrites formations for the second one.

The model presented in [Chopard *et* Luthi 1994] is an archetype of those fictitious world. Instead of modeling the governing differential equations, instead of attempting to model the details of motion and reaction of chemical species into a solvent, one defines a new world, with very simplified rules: discrete particles, from species A and B, move randomly on a regular grid and transform when particle of both species encounter on a site.

Although model details are far from reality details (particles are discrete quantitites moving synchrounously on a discrete lattice and reaction process is largely simplified), simulations produces a global complex behavior very close to the real one, a success which was first difficult to reach with other numerical approaches.

1.2.3 ... and many others complex systems

Many systems (if not any, according to some peoples) have been addressed by similar methods. Results have been published in a vey wide range of applications, including traffic, ants, forest fires, radio-waves propagations, excitable medias, turing patterns models, competition among smokers and non-smokers [Galam *et al.* 1998] and many more [Chopard *et* Droz 1998].

1.3 The present work

The present thesis shows how these ideas can be applied to model snow transport and deposition by wind, which mixed two major components:

1. a fluid, composed of virtual particles interacting on a lattice,
2. solid particles, eroded, transported and deposited under the action of the fluid.

In a first part, we will present a fluid model, from a basic discrete cellular automata up to a very efficient model modeling high Reynolds number flows. Although it has been proven to be theorytically correct, we will also validate the model from a practical point of view, confronting it with results of the literature.

In a second part, we will present the original part of the thesis, *i.e.* the inclusion of particles on the top of the fluid model. In the same manner, we will show results within a very wide range of space scales, from little ripples up to deposits around mountain crests.

In the last part, we will focus on the implementation of such an application in Fortran 90 and overall on its parallelization with message passing libraries.

Last but not least, several appendixes present original complements to the current works: a multiparticle fluid model, an interactive simulation environment, a short guide to the visualization software *AVS/Express*, and a home made benchmarks of MPI on four machine architectures.

This whole work includes ideas from the literature (mainly for the flui part) and a large original work, both in the usage of such fluid models for real-life turbulent apllications and the incorporation of the solid particles to predict deposits. Moreover, such simulations need large computational resources, therefore, the use of different powerful parallel computers have allowed short computing times showing how such machines can change the life of the numericist.

Chapter 2

The Fluid Model

To model the fluid, we have adopted the so-called *lattice gas* technique, and more precisely the *lattice BGK subgrid* model. To introduce this model, we will briefly step through *cellular automata*, *lattice gas automata* and their first level "mutation" to general *lattice Boltzmann* models.

Once the model has been described, we will focus on a set of experiments, confronting simulations with results from the literature.

2.1 Lattice gas: a short introduction

As introduced in section 1.2, lattice models transpose reality into a fictitious world, where the governing rules are greatly simplified. Before focusing on the description of lattice gaz models, we briefly point out three scale levels of the description of real phenomenon:

1. the macroscopic,
2. the microscopic,
3. and the mesoscopic scales.

The macroscopic scale

To model a natural phenomenon (fluid, population evolution or radio-waves propagation), scientists have tried to extract equations from their observations (Navier-Stokes, Lotka-Volterra or Maxwell equations), and later have used these equations for a better understanding of the phenomenon. However, when one wishes to look closer and needs to incorporate new elements, these equations quickly become very complicated.

Later, when computers have proven themselves to be appealling modelling tools, the most popular idea to simulate the natural phenomenon was to numerically model the corresponding set of equations (thus numerically modelling a mathematical model).

The microscopic scale

On the other hand, instead of modelling coarse grain quantities through macroscopic equations (*e.g.* Lotka-Volterra like equations for prey/predators populations, considering paramaters such as density of both species, rate of birth etc.), one can try to simulate individual behavior of every component of the phenomenon (each prey or predator has its own location - if one of each meet, the prey dies -, motion, behavior, reproduction cycle).

The mesoscopic scale

To model a fluid one can either use the Navier-Stokes equations through a differential equation solver, or use an atomic representation through a molecular dynamic system (every molecule is represented and complex collisions are taken into account).

In between, one can imagine to represent the fluid by a set of basic "particles", evolving in a fictitious world, reacting with simplified and relevant rules [1]. Although this representation is far away from the richness of reality, it may be enough to recover complex features of the natural phenomenon.

Both the macroscopic and the microscopic scales are very complicated to simulate, while the mesoscopic scale provides an attractive level of description, as they are ideally suited for numerical simulations. Moreover, instead of being obsessed by reproducing reality in the closest manner possible, one builds his own world, with his own set of rules governing elements which have not the pretention of individually representing their equivalent in reality. However, if the rules catch the key ingredients of the pheomenon, the global behavior of the system can recover complex results.

Cellular automata, described in the following subsection, are representative of those mesoscopic scale models.

2.1.1 Cellular automata

Cellular automata (CA) are fictitious physical system, where [Chopard *et al.* 1998a]:

- space domain is a regular lattice (Γ); each lattice node $\mathbf{r}$ is linked to q neighbors $\{\mathbf{r}+\mathbf{c}_i\}_{i=0,q-1}$;
- the cell states $N(\mathbf{r},t)$ belong to a finite set of values;

[1] *"Everything should be made as simple as possible but not simpler"* A. Einstein, cited in [Chopard *et* Droz 1998]

- evolution is synchronous, discrete in time and update depends only on the local state and those of the nearest neigbors following a rule $\Re$:

$$N(\mathbf{r}, t+1) = \Re(N(\mathbf{r}, t), N(\mathbf{r}+\mathbf{c}_0, t) \dots N(\mathbf{r}+\mathbf{c}_{q-1}, t)) \qquad (2.1)$$

The game of life

This famous cellular automata was proposed by the mathematician John Conway in the seventies [Gardner 1970]: on a 2D lattice, where each node is linked to eight neighbors (north, north-east, east, south-east, south, south-west, west and north-west), a cell can be either dead (0) or alive (1):

- a dead cell surrounded by exactly 3 alive cells regains life,
- a living cell surrounded by less than two cells alive cells dies of isolation,
- a living cell surrounded by more than three cells alive neighbors dies of overcrowdness.

Let $\sigma_N(\mathbf{r}, t)$ (abbreviated σ_N) be the sum of the neighboring alive cells of site $\mathbf{r}$ at time t:

$$\sigma_N(\mathbf{r}, t) = \sum_{i=0}^{3} N(\mathbf{r}+\mathbf{c}_i, t)$$

The evolution rule can therefore be written as:

$$N(\mathbf{r}, t+1) = \begin{cases} 1 & \text{if } \sigma_N(\mathbf{r}, t) = 3 \\ N(\mathbf{r}, t) & \text{if } \sigma_N(\mathbf{r}, t) = 2 \\ 0 & \text{if } \sigma_N(\mathbf{r}, t) < 2 \\ 0 & \text{if } \sigma_N(\mathbf{r}, t) > 3 \end{cases}$$

Even if this rules can be seen as a toy model at first glance, a closer look shows how this CA can exhibit complex behaviors, and even be able to reproduce any computational process. However, although it is very rich from the theoritical point of view, it is far from representing a real animal “game of life process”.

We will focus in the following paragraphs on CA designed to model a fluid, namely the lattice gas automata (LGA).

HPP

The earliest LGA is the HPP model (Hardy, Pomeau and de Pazzis [Hardy *et al.* 1973]), where a very simplified molecular dynamics is described: Boolean particles move synchronously on a square lattice (four neighbors); only one particle can travel at once on a lattice link (exclusion principle); a collision takes place when more than one particle enter a site at the same time step, accordingly to the rule described in figure 2.1.

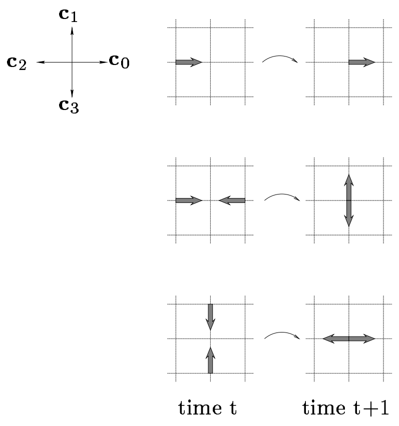


Figure 2.1: HPP collision. Only when two particles enter a site with opposite directions, they are deflected perpendiculary; all the other configurations remain identical.

From a more practical point of view, four vectors $\{\mathbf{c}_i\}_{i=0,1,2,3}$ describe the lattice Γ. The local state of a cell consists of 4 Boolean values $\{F_i(\mathbf{r},t)\}_{i=0,1,2,3}$ where $F_i(\mathbf{r},t) = 0/1$ indicates the absence/presence of a particle travelling in direction $\mathbf{c}_i$ on site $\mathbf{r}$, at time t.

If only one particle, with velocity $\mathbf{c}_i$, enters the site at time t, it is streamed to $\mathbf{r}+\mathbf{c}_i$ at time $t+1$; if two particles enter a site with opposite velocities, they collide and are deflected in the perpendicular direction; when more particles enter the site, the situation remains unchanged and particles are only streamed. This collision rule can be written as:

$$\begin{aligned} F_i(\mathbf{r}+\mathbf{c}_i,t+1) &= F_i(\mathbf{r},t) \\ &- F_i(\mathbf{r},t)F_{i+2}(\mathbf{r},t)(1-F_{i+1}(\mathbf{r},t))(1-F_{i+3}(\mathbf{r},t)) \\ &+ F_{i+1}(\mathbf{r},t)F_{i+3}(\mathbf{r},t)(1-F_i(\mathbf{r},t))(1-F_{i+2}(\mathbf{r},t)) \end{aligned} \tag{2.2}$$

where all indexes are taken modulo 4, the first term is for the *statu quo*, the second one when only one particle in direction i and another in the opposite direction $(i+2)$ enter the site, thus F_i vanishes, and the last one when only two particles are entering the site in direction perpendicular to $\mathbf{c}_i$ ($i+1$ and $i+3$), thus F_i becomes 1.

Basic definitions

It is therefore very natural to define the number of particle:

$$\rho(\mathbf{r},t) = \sum_{i=0}^{q_f-1} F_i(\mathbf{r},t) \tag{2.3}$$

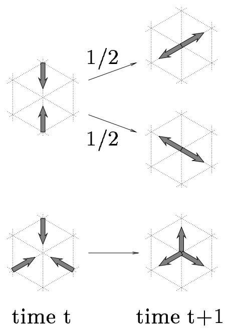


Figure 2.2: FHP collision, on an hexagonal lattice. When only two particles enter a site with opposite directions, they are deflected with a probability 1/2 of 60 or −60 degres. When three particles enter the site in a way that the momentum is zero, their velocity are inversed.

and the local momentum, *i.e.* the number of particles multiplied by the velocity:

$$\mathbf{J}(\mathbf{r},t) = \sum_{i=0}^{q_f-1} F_i(\mathbf{r},t)\mathbf{c}_i = \rho(\mathbf{r},t)\mathbf{u}(\mathbf{r},t) \tag{2.4}$$

where q_f is the number of neighbors for the fluid cellular automata. The usual density and momentum are usually obtained averaging these terms.

FHP, FHP-III

Although the HPP formulation is elegant, it cannot be considered as a fluid model (*i.e.* satisfying the Navier-Stokes equations), because of some intrinsic anisotropy of the lattice preventing a coherent definition of the momentum tensor. However, an hexagonal lattice ($q_f = 6$) can satisfy this isotropy assumption, and collision rules were imagined by Frisch, Hasslacher and Pomeau (FHP) [Frisch *et al.* 1986] (see figure 2.2).

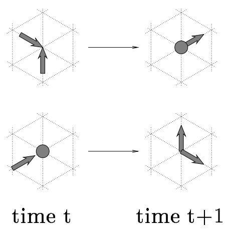


Figure 2.3: FHP-III collisions involving a rest particle (the others are the same as for the FHP model). Density and momentum are conserved, but not the energy. However, it is conserved on average over the collisions.

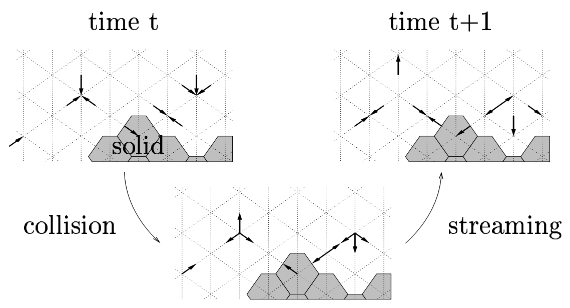


Figure 2.4: FHP evolution step, decomposed in two stages: a) collision, b) information streaming to the neighboring cells.

To get richer collisions, a particle at rest is added in the FHP-III set of rules. In this case, q_f is considered to be 7, and one of the indexes is used for the rest particles. Later, even multi-speed models have been proposed in the litterature [Doolen 1990].

One can notice that, for each rule, mass and momentum, as defined in equations (2.3) and (2.4), are conserved. This, together with some lattice isotropy assumption, ensures that the system satisfies the Navier Stokes equations, up to some non Galilean invariance correction term [Chopard *et al.* 1998a, Qian *et al.* 1996a]; the latter problem will vanish when using a Lattice Boltzmann model, *i.e.* in the practical model used in the framework of this thesis.

However, in these models, viscosity depends on the collision rule and the average density of particles. Thus it exists only in a restrictive range and cannot be tuned to reach interessant values (in the aim of modelling turbulent situations).

Solid as boundary conditions

As the fluid is described by CA particles, evolving synchronously on a regular lattice, the interaction with solid can be handled in a very efficient and elegant way: instead of interacting normally according to the rules described above, a particle entering a solid site simply bounce back (inversing its velocity).

To be or not to be solid is considered as a new Boolean variable, one more in the description of the local cell state, and the bounce back condition can be incorporated *as is* in the CA set of rules.

Implementation

The cellular automata implementation will be more thoroughly discussed in chapter 4, however a CA evolution step can be summarized as a parallel (synchronously on all the lattice sites) two stage mechanism:

1. collision computations, based on the local information; as the local situation can be described by q_f bits, a *lookup table* with 2^{q_f} entries returning a q_f bit

integer can store all the possible evolutions; this lookup table can computed at once and the collision step becomes only a memory load rather than a expression computation;

2. streaming of information to the neighboring cells.

Such an evolution step, for the FHP rule, is displayed in figure 2.4.

2.1.2 The lattice Boltzmann method

Cellular automata are an idealization of a physical system, however, several inconveniences are intrinsically linked to their discrete nature:

- the high statistical noise requires large system averages,
- lack of possiblities for the fine tuning of the input conditions,

Lattice Boltzmann models offer a solution to these problems: instead of dealing with the Boolean presence/absence of a particle on a given lattice link, they consider the *probability* of having the particle on this link. Instead of $F_i \in \{0, 1\}$, they consider f_i ranging continuously in $[0, 1]$.

If N-body correlations are neglected, the collision rule expressed for discrete quantities (such as in equation (2.2) for HPP) can be applied straightforwardly to the average values [McNamara *et* Zanetti 1988]. Local density and momentum can be computed in the same manner as in equations (2.3) and (2.4), but are now continuous, lowering drastically the statistical noise.

However, the price to pay for this major improvement are numerical instabilities and the loss of the N-body correlations, which can easily be neglected in most of the fluid simulation problems, but become important when distribution fluctuations are essential. To merge both techniques, we have imagined the technique described in annex A.

Model limitations

Although the HPP collision term (equation (2.2)) is rather simple, it becomes more complex for the FHP-III model (6 neighbors and a rest particle):

$$
\begin{array}{rl}
 & F_i(t+1, r+\vec{c_i}) = F_i \\
+ & \frac{1}{2}\overline{F_i}\,\overline{F_{i+3}}\,(F_{i+1}F_{i+4}\overline{F_{i+2}}\,\overline{F_{i+5}} + F_{i+2}F_{i+5}\overline{F_{i+1}}\,\overline{F_{i+4}}) \\
 & \quad - F_iF_{i+3}\overline{F_{i+1}}\,\overline{F_{i+2}}\,\overline{F_{i+4}}\,\overline{F_{i+5}} \\
+ & F_{rest}\overline{F_{i+3}}\,\overline{F_{i+4}}\,(\overline{F_i}\,(F_{i+1}\overline{F_{i+2}}\,\overline{F_{i+5}} + F_{i+5}\overline{F_{i+1}}\,\overline{F_{i+2}}) \\
 & \quad - F_i\overline{F_{i+1}}\,\overline{F_{i+2}}\,\overline{F_{i+5}}) \\
+ & \overline{F_{i+3}}\,\overline{F_{rest}}\,(F_{i+1}F_{i+5}\overline{F_i}\,\overline{F_{i+2}}\,\overline{F_{i+4}} - F_i\overline{F_{i+1}}\,\overline{F_{i+5}}\,(F_{i+2}\overline{F_{i+4}} + F_{i+4}\overline{F_{i+2}})) \\
+ & F_{i+1}F_{i+3}F_{i+5}\overline{F_i}\,\overline{F_{i+2}}\,\overline{F_{i+4}} - F_iF_{i+2}F_{i+4}\overline{F_{i+1}}\,\overline{F_{i+3}}\,\overline{F_{i+5}} \\
+ & F_{rest}(\overline{F_i}\,(F_{i+1}F_{i+5}\overline{F_{i+2}}\,\overline{F_{i+3}}\,\overline{F_{i+4}} \\
 & \quad + \frac{1}{2}(F_{i+3}\overline{F_{i+2}}\,\overline{F_{i+4}}\,(F_{i+5}\overline{F_{i+1}} + F_{i+1}\overline{F_{i+5}}))) \\
 & \quad - \frac{1}{2}F_i\overline{F_{i+1}}\,\overline{F_{i+3}}\,\overline{F_{i+5}}\,(F_{i+4}\overline{F_{i+2}} + F_{i+2}\overline{F_{i+4}})) \\
+ & \overline{F_{rest}}\,(\overline{F_i}\,(F_{i+1}F_{i+2}F_{i+5}\overline{F_{i+3}}\,\overline{F_{i+4}} + \frac{1}{2}F_{i+2}F_{i+4}F_{i+5}\overline{F_{i+1}}\,\overline{F_{i+3}}) \\
 & \quad - F_i(\frac{1}{2}\overline{F_{i+2}}\,(F_{i+1}F_{i+4}\overline{F_{i+3}}\,\overline{F_{i+5}} + F_{i+3}F_{i+5}\overline{F_{i+1}}\,\overline{F_{i+4}}) \\
 & \quad + F_{i+2}F_{i+3}\overline{F_{i+1}}\,\overline{F_{i+4}}\,\overline{F_{i+5}})) \\
+ & \overline{F_{rest}}\,(\overline{F_i}\,(F_{i+1}F_{i+4}F_{i+5}\overline{F_{i+2}}\,\overline{F_{i+3}} + \frac{1}{2}F_{i+1}F_{i+2}F_{i+4}\overline{F_{i+3}}\,\overline{F_{i+5}}) \\
 & \quad - F_i(F_{i+3}F_{i+4}\overline{F_{i+1}}\,\overline{F_{i+2}}\,\overline{F_{i+5}} \\
 & \quad + \frac{1}{2}\overline{F_{i+4}}\,(F_{i+1}F_{i+3}\overline{F_{i+2}}\,\overline{F_{i+5}} + F_{i+2}F_{i+5}\overline{F_{i+1}}\,\overline{F_{i+3}}))) \\
+ & F_{rest}(\overline{F_i}\,(F_{i+1}F_{i+2}F_{i+5}\overline{F_{i+3}}\,\overline{F_{i+4}} \\
 & \quad + \frac{1}{2}F_{i+4}(F_{i+2}F_{i+5}\overline{F_{i+1}}\,\overline{F_{i+3}} + F_{i+1}F_{i+3}\overline{F_{i+2}}\,\overline{F_{i+5}})) \\
 & \quad - F_iF_{i+3}\overline{F_{i+1}}\,\overline{F_{i+4}}\,(F_{i+2}\overline{F_{i+5}} + \frac{1}{2}F_{i+5}\overline{F_{i+2}})) \\
+ & F_{rest}(\overline{F_i}\,(F_{i+1}F_{i+4}F_{i+5}\overline{F_{i+2}}\,\overline{F_{i+3}} \\
 & \quad + \frac{1}{2}F_{i+2}(F_{i+3}F_{i+5}\overline{F_{i+1}}\,\overline{F_{i+4}} + F_{i+1}F_{i+4}\overline{F_{i+3}}\,\overline{F_{i+5}})) \\
 & \quad - F_iF_{i+3}\overline{F_{i+2}}\,\overline{F_{i+5}}\,(F_{i+4}\overline{F_{i+1}} + \frac{1}{2}F_{i+1}\overline{F_{i+4}})) \\
+ & F_{i+1}F_{i+2}F_{i+4}F_{i+5}\overline{F_i}\,\overline{F_{i+3}} \\
 & \quad - \frac{1}{2}F_iF_{i+3}(F_{i+1}F_{i+4}\overline{F_{i+2}}\,\overline{F_{i+5}} + F_{i+2}F_{i+5}\overline{F_{i+1}}\,\overline{F_{i+4}}) \\
+ & \overline{F_{rest}}\,(\frac{1}{2}\overline{F_i}\,F_{i+1}F_{i+3}F_{i+5}(F_{i+4}\overline{F_{i+2}} + F_{i+2}\overline{F_{i+4}}) \\
 & \quad - F_iF_{i+2}F_{i+4}(\frac{1}{2}\overline{F_{i+3}}\,(F_{i+1}\overline{F_{i+5}} + F_{i+5}\overline{F_{i+1}}) + F_{i+3}\overline{F_{i+1}}\,\overline{F_{i+5}})) \\
+ & \overline{F_{rest}}\,F_{i+2}F_{i+3}F_{i+4}(F_{i+1}F_{i+5}\overline{F_i} - F_i(F_{i+1}\overline{F_{i+5}} + F_{i+5}\overline{F_{i+1}})) \\
+ & F_{rest}F_{i+3}(F_{i+1}F_{i+5}\overline{F_i}\,(F_{i+4}\overline{F_{i+2}} + F_{i+2}\overline{F_{i+4}}) - F_iF_{i+2}F_{i+4}\overline{F_{i+1}}\,\overline{F_{i+5}})
\end{array}
$$

where F_i states for $F_i(\mathbf{r}, t)$ and $\overline{F_i}$ for $(1 - F_i)$. The evolution rule for the rest particle density can also be expressed by such a cute little formula.

The complexity of this term expresses clearly the exponential dependence of the collision rule on the number of particle. If it can still be plausible to compute a FHP-III model directly from the evolution term, it would be utopian to undertake 3D simulations (inspired from 24 particles model).

Moreover, the model is strongly related to the underlying cellular automata, in the sense that the viscosity is directly linked to the collision rule and thereore hardly tunable.

Fortunaltely, the collision term can be drastically simplified and the lack of independent viscosity solved, as described in the following section.

2.2 Lattice BGK subgrid model

2.2.1 Linearizing the collision term

To achieve more efficient simulations (such as faster computations or higher Reynolds numbers), a major limit of the pre-cited methods is, on the one hand, the exponential dependence of the complexity with the number of lattice links and, on the other hand, the absence of free parameter to tune the viscosity (produced by the model geometry, the size of the lattice and the collision rule).

Breaking both these barriers was therefore a key challenge to make Lattice Gas a useful tool from a practical point of view. This evolution has been clearly described in [Qian *et al.* 1996a, Chopard *et* Droz 1998] and is very briefly summarized in this section.

Split of the local distribution in equilibrium + non-equilibrium parts

A major step in the improvement of lattice gas models was to decompose the density distribution as an equilibrium distribution term plus a non-equilibrium fluctuation term [Higuera *et al.* 1989a]:

$$f_i = f_i^{eq} + f_i^{neq} \qquad \forall i \in \{0, q_f\} \tag{2.5}$$

where the local equilibria $\{f_i^{eq}\}$ depends on the local density ρ and velocity $\mathbf{u}$ and is given by the Fermi-Dirac distribution (obeying the exclusion principle, *i.e.* no more than one particle per lattice link) [Frisch *et al.* 1987]:

$$f_i^{eq} = \frac{1}{1 + \exp(-A - \vec{B} \cdot \mathbf{c}_i)} \tag{2.6}$$

where A and $\vec{B}$ are quantities such that local density and velocity are conserved.

"Taylorization" of the equilibrium distribution term

In order to carry out the calculations, (2.6) is Taylor expanded:

$$f_i^{eq} = \rho t_p \left[1 + \frac{\mathbf{c}_{i\alpha}\mathbf{u}_\alpha}{\mathbf{c_s}^2} + \frac{1}{2}\left(\frac{\mathbf{c}_{i\alpha}\mathbf{u}_\alpha}{\mathbf{c_s}^2}\right)^2 - \frac{\mathbf{u}_\alpha\mathbf{u}_\alpha}{2\mathbf{c_s}^2} \right] \tag{2.7}$$

where summations over repeated spatial indices are assumed and $\mathbf{v}_\alpha$ is the component of $\mathbf{v}$ in dimension α, $\mathbf{c_s}$ the speed of sound and t_p weights depending on the amplitude of the vector $\mathbf{c}_i$, such that $p = p(i) = \| \mathbf{c}_i \|^2$ (*i.e.* in 2D square model, t_0 weights the rest particles, t_1 the particles moving horizontally and vertically, and t_2 the particles moving along the diagonals). These weights are model dependent and must fulfill basic conservations and isotropy requirements [Qian *et al.* 1996a, p. 208]. A set of parameters is given in table 2.1 for different lattice models.

The collision term

Therefore, we can state the evolution as:

$$\begin{aligned} f_i(\mathbf{r}+\mathbf{c}_i, t+1) &= f_i(\mathbf{r}, t) + \Omega_i(f_i(\mathbf{r}, t)) \\ &= f_i(\mathbf{r}, t) + \Omega_i(f_i^{eq}(\rho(\mathbf{r}, t), \mathbf{u}(\mathbf{r}, t)) + f_i^{neq}(\mathbf{r}, t)) \\ &= f_i(\mathbf{r}, t) + \Omega_i(f_i^{neq}(\mathbf{r}, t)) \\ &= f_i(\mathbf{r}, t) + \Omega_i(f^{eq}(\rho(\mathbf{r}, t), \mathbf{u}(\mathbf{r}, t)) - f(\mathbf{r}, t) \end{aligned} \tag{2.8}$$

The first idea was to express the collision operator Ω as a matrix in the so-called *quasi-linear lattice Boltzmann equation* [Higuera *et al.* 1989a], reducing the complexity from 2^{q_f} to ${q_f}^2$. It was improved, stepping away from the underlying discrete cellular automata and reaching lower viscosities with the *lattice Boltzmann equation with enhanced collisions* [Higuera *et al.* 1989b].

Meanwhile, the collision operator complexity was further decreased to the order of q_f with the advent of the BGK models.

2.2.2 The BGK model

In a different context, Bathnagar, Gross and Krook used a simple relaxation term to model the collision operator in a standard Boltzmann equation [Bathnagar *et al.* 1954]. Their key idea has been adapted to lattice Boltzmann model: the collision term Ω is reduced to its simplest form, and no explicit underlying collision model is incorporated into it any more. Equation (2.8) can therefore be simplified :

$$f_i(\mathbf{r}+\mathbf{c}_i, t+1) = f_i(\mathbf{r}, t) + \frac{1}{\tau}\left(f_i^{eq}(\rho(\mathbf{r}, t), \mathbf{u}(\mathbf{r}, t)) - f_i(\mathbf{r}, t)\right) \tag{2.9}$$

where τ is the *relaxation time*, a free parameter directly related to the kinematic viscosity ν, through [Qian *et al.* 1996a]:

$$\nu = \frac{{\mathbf{c_s}}^2}{2}(2\tau - 1) \tag{2.10}$$

Therefore, τ must be greater than 1/2 and the viscosity should tend towards 0 as τ tends towards 1/2, but numerical instabilities appear when τ is too low.

Fortunately, human genius has, once again, saved this hopeless situation ...

	t_0	t_1	t_2	t_3	t_4	${\mathbf{c_s}}^2$	C_2	C_4	
$D1Q3$	2/3	1/6	0	0	0	1/3	1/3	1/9	only the nearest neighbors
$D1Q5$	1/2	1/6	0	0	1/12	1	1	1	neighbors at distance 1 and 2
$D2Q7$	1/2	1/12	0	0	0	1/4	1/4	1/16	hexagonal
$D2Q9$	4/9	1/9	1/36	0	0	1/3	1/3	1/9	square + diagonals
$D3Q15$	2/9	1/9	0	1/72	0	1/3	1/3	1/9	cubic, middle of faces + corners
$D3Q19$	1/3	1/18	1/36	0	0	1/3	1/3	1/9	cubic, middle of faces and edges
$D4Q25$	1/3	0	1/36	0	0	1/3	1/3	1/9	4D regular paving

Table 2.1: The $DdQq$ models, where: d is the dimension; q the number of particle velocities; t_p is the weight associated with directions $\mathbf{c}_i$ where $p = \| \mathbf{c}_i \|^2$; $\mathbf{c_s}$ is the speed of sound; C_2 and C_4 are defined to fullfill $\sum_{i=1}^{q} t_{\|\mathbf{c}_i\|} \mathbf{c}_{i\alpha} \mathbf{c}_{i\beta} = C_2 \delta_{\alpha\beta}$ and $\sum_{i=1}^{q} t_{\|\mathbf{c}_i\|} \mathbf{c}_{i\alpha} \mathbf{c}_{i\beta} \mathbf{c}_{i\delta} \mathbf{c}_{i\gamma} = C_4 (\delta_{\alpha\beta}\delta_{\gamma\delta} + \delta_{\alpha\gamma}\delta_{\beta\delta} + \delta_{\alpha\gamma}\delta_{\beta\delta})$, $\forall \alpha, \beta, \gamma$ and δ, where $\delta_{\alpha\beta}$ is the Kronecker function (the fulfilment of both conditions imposes constraints on the choice of the lattice and its direction weights).

2.2.3 Subgrid model

To model higher Reynolds number, *i.e.* more turbulent flows, with the previous model, two actions can be undertaken:

- increasing $\mathbf{u}_{entry}$, thus worsening the incompressibility error,
- lowering τ, but the simulation numerically blows up,
- increasing the lattice size, needing more memory and CPU power, but the budget will also blow up.

Reading [Hou 1995] (summarized, and focused on the 2D cavity flow problems in [Hou *et al.* 1996]) shows a way to cleverly break this new barrier. This approach is summarized in the following paragraphs.

Large Eddy Simulation and eddy viscosity

The idea is to adapt the Large Eddy Simulation (*LES*): modelling only the large eddies (larger than the lattice resolution), and trying to extrapolate what is happenning at lower resolution scales (in fact modelling the effect of the unresolved scales on the effective ones). This method is briefly presented here.

Starting from the basic scale level, with a defined physical variable w (w can either be f_i, $\mathbf{u}$ or ρ), we first define a coarser scale (which will be the computations scale level), and express $\overline{w}$ at this new level:

$$\overline{w(\mathbf{r})} = \int w(\mathbf{r}')G\left(\mathbf{r} - \mathbf{r}'\right) d\mathbf{r}' \tag{2.11}$$

where G is a given spatial filter (averaging w, for example, on a box).

It has been shown by [Hou 1995] how this small scale extrapolation can be included as a spatial dependent kinematic viscosity $\overline{\nu}$ taking into account the originally given viscosity ν_0 plus a so-called *eddy viscosity* ν_t:

$$\overline{\nu} = \nu_0 + \nu_t \tag{2.12}$$

This eddy viscosity is sometimes seen as an empirical add-on without “clear link with the underlying physics”. However it works and many approaches have emerged to compute this add-on. Among them, we present in the current work the Smagorinski eddy viscosity model:

$$\nu_t = C_{smago}|S| \tag{2.13}$$

where $|S| = \sqrt{2S_{\alpha\beta}S_{\alpha\beta}}$ is the magnitude of the strain-rate tensor $S_{\alpha\beta}$ and C_{smago} an external parameter called the Smagorinski constant (we shall discuss its values later).

As kinematic viscosity and relaxation time are linked through equation (2.10), an *eddy relaxation time* τ_t can also be considered:

$$\overline{\tau} = \frac{\overline{\nu}}{\mathbf{c_s}^2} - \frac{1}{2} = \frac{\nu_0 + \nu_t}{\mathbf{c_s}^2} - \frac{1}{2} = \tau_0 + \underbrace{\frac{\nu_t}{\mathbf{c_s}^2}}_{\tau_t} \tag{2.14}$$

The strain-rate tensor

The strain-rate tensor is traditionally computed as a finite difference directly from its definition:

$$S_{\alpha\beta} = \frac{1}{2}\left(\frac{\partial \overline{\mathbf{u}}_\alpha}{\partial x_\beta} + \frac{\partial \overline{\mathbf{u}}_\beta}{\partial x_\alpha}\right) \tag{2.15}$$

However, in the BGK model, when high order velocity effects and incompressibility errors are neglected, $S_{\alpha\beta}$ can be deduced from the flux-tensor momentum [Chopard *et* Droz 1998]:

$$\Pi_{\alpha\beta} = -\tau\rho\frac{C_4}{C_2}\left(\frac{\partial \overline{\mathbf{u}}_\alpha}{\partial x_\beta} + \frac{\partial \overline{\mathbf{u}}_\beta}{\partial x_\alpha}\right) \Rightarrow S_{\alpha\beta} = -\frac{1}{2\tau\rho}\frac{C_2}{C_4}\Pi_{\alpha\beta} \tag{2.16}$$

where C_2 and C_4 are lattice dependent:

$$\begin{gathered}\sum_{i=1}^{q_f} t_{\|\mathbf{c}_i\|}\mathbf{c}_{i\alpha}\mathbf{c}_{i\beta} = C_2\delta_{\alpha\beta} \\ \sum_{i=1}^{q_f} t_{\|\mathbf{c}_i\|}\mathbf{c}_{i\alpha}\mathbf{c}_{i\beta}\mathbf{c}_{i\delta}\mathbf{c}_{i\gamma} = C_4(\delta_{\alpha\beta}\delta_{\gamma\delta} + \delta_{\alpha\gamma}\delta_{\beta\delta} + \delta_{\alpha\gamma}\delta_{\beta\delta}) \\ \forall \alpha, \beta, \gamma, \delta\end{gathered} \tag{2.17}$$

where $\delta_{\alpha\beta}$ is the Kronecker function. The definition of C_2 and C_4 is only coherent for isotropic lattices (they are inconsistent, for example, for a $D2Q4$ lattice).

A step by step method

Equations (2.14) and (2.16) lead to a local algorithm for the computation of $\overline{\tau}$:

a) the non-equilibrium momentum flux tensor[2]:

$$\Pi_{\alpha\beta} = \sum_i \mathbf{c}_{i\alpha}\mathbf{c}_{i\beta}\left(\overline{f_i} - \overline{f_i^{eq}}\right) \tag{2.18}$$

[2]**Warning:** in [Hou 1995] and [Hou *et al.* 1996], there is a misprint in the definition of $\Pi_{\alpha\beta}$ wrongly stated as $\Pi_{ij} = \mathbf{c}_{i\alpha}\mathbf{c}_{j\alpha}\left(\overline{f_i} - \overline{f_i^{eq}}\right)$. This definition even contradicts the assumption that the non-equilibrium momentum flux tensor is linearly related to the strain-rate tensor, which is defined without reference to the lattice notation in (2.15). When directly applying the cited algorithm, the renormalization by C_{smago} hides the quantitative error and the qualitative results looks correct; however, in 3D, the error is emphasized, even in simple fluid velocity profiles of an open channel.

b) its scalar product $Q = \Pi_{\alpha\beta}\Pi_{\alpha\beta}$,where summation over repeated indices are still assumed,

c) the link to $|S|$ is done, summing over the spatial indices relation (2.16):

$$|S| = \frac{C_2}{C_4}\frac{1}{2\rho(\tau_0+\tau_t)}\sqrt{Q} \tag{2.19}$$

$$\Rightarrow \nu_t = \mathbf{c_s}^2\tau_t = C_{smago}\frac{C_2}{C_4}\frac{1}{2\rho(\tau_0+\tau_t)}\sqrt{Q} \tag{2.20}$$

d) solving this second order equation for τ_t (> 0), one obtains an explicit subgrid relaxation time:

$$\overline{\tau} = \frac{1}{2}\left(\sqrt{\tau_0^2 + C_{smago}\frac{2C_2}{C_4\mathbf{c_s}^2}\frac{\sqrt{Q}}{\rho}} + \tau_0\right) \tag{2.21}$$

And the evolution equation (2.9) becomes:

$$\overline{f_i}(\mathbf{r}+\mathbf{c}_i, t+1) = \overline{f_i}(\mathbf{r},t) + \frac{1}{\overline{\tau}}\left(\overline{f_i^{eq}}(\rho(\mathbf{r},t),\mathbf{u}(\mathbf{r},t)) - \overline{f_i}(\mathbf{r},t)\right) \tag{2.22}$$

For simplicity, the $\_$ symbols can be removed, and the basic BGK scheme is only modified by the extrapolation (2.21). Moreover, from a practical point of view, it is interessant to fix $\tau_0 = 0.5$ in order to reach the lowest viscosity possible (as it will be the case from now on in this thesis, when the subgrid model is involved).

The global program for the fluid collision step is displayed in algorithm 2.2.1, which can be directly translated in a classic programming language.

Why the Smagorinski model avoids the numerical blow up: an intuitive explanation

Numerical blow up is generated where the f_i's reach negative values, *i.e.* where the strain is too high (close to an obstacle, by example) and the relaxation time τ too small. These negative values could be avoided by increasing τ, but this is in contradiction with the quest for low viscosity. The Smagorinski model presented in this section precisely increase the relaxation only where the strain is large, thus killing the blowing up germs before they can develop.

The value of C_{smago} is therefore important, as, on the one hand, a too high C_{smago} will generate a too high relaxation time with no chance of decreasing the viscosity; on the other hand, a too small C_{smago} will not prevent the system from blowing up. The right value to obtain the most developed turbulence depends mainly on the flow configuration, and must be set empirically. This empiricism is the main drawback of the Smagorinski model and can be solved using other subgrid models.

Algorithm 2.2.1 global evolution code for the subgrid BGK model, where the f_i''s are the evolution of the distribution f_i's (the streaming stage consists therefore only of $f_i(\mathbf{r}+\mathbf{c}_i, t+1) \leftarrow f_i'(\mathbf{r}, t)$). This algorithm can of course be optimized in some manners, but its intrinsic simplicity (a couple of dozens code lines for a 2D/3D fluid solver!) and its efficiency are two (other) strong advantages of the method.

```
//temporary variables are
//counters: $i, j, \alpha$
//$\Theta, \delta f, Q, \Pi$

//$f_i^{eq}$ evaluation
$f_0^{eq} \leftarrow \rho t_0(1 - \|\mathbf{u}\|^2/(2\mathbf{c_s}^2))$
do $i = 0, q_f - 1$
   !$\Theta$ contains $\mathbf{c}_i.\mathbf{u}/\mathbf{c_s}^2$
   $\Theta \leftarrow 0$
   do $\alpha = 1, d$
      $\Theta \leftarrow \Theta + \mathbf{c}_{i\alpha}\mathbf{u}_\alpha$
   enddo
   $\Theta \leftarrow \Theta/\mathbf{c_s}^2$
   $f_i^{eq} \leftarrow \rho t_{\|\mathbf{c}_i\|}(1 + \Theta(1 + \Theta/2) - \|\mathbf{u}\|^2/(2\mathbf{c_s}^2)$
enddo

//$\overline{\tau}$ evaluation
$Q \leftarrow \quad 0$
do $i = 0, q_f - 1$
   $\delta f_i \leftarrow f_i - f_i^{eq}$
enddo
do $\alpha = 1, d$
   do $\beta = 1, d$
      $\Pi \leftarrow \quad 0$
      do $j = 0, q_f - 1$
         $\Pi \leftarrow \Pi + \mathbf{c}_{i\alpha}\mathbf{c}_{i\beta}\delta f_i$
      enddo
      $Q \leftarrow \quad Q + \Pi^2$
   enddo
enddo
$\overline{\tau} \leftarrow \tau + \left(\sqrt{\tau^2 + C_{smago}(2C_2/(C_4\mathbf{c_s}^2))(\sqrt{Q}/\rho)} - \tau\right)/2$

//and finally
do $i = 0, q_f - 1$
   $f_i$'$\leftarrow (1 - 1/\overline{\tau})f_i + 1/\overline{\tau} f_i^{eq}$
enddo
```

2.3 Fluid validation

In this section we will present numerical experiments we have performed to validate the BGK model in order to reproduce turbulent flows.

We shall therefore compare numerical results with situations that may be found in the litterature, either from a theoretical or an experimental point of view. All simulations have been run with a 3D ($D3Q19$) model, but over thin (small $y-$dimension) domains as results in the litterature are mainly 2D view of a 3D reality (the shape of the obstacle usually don't change along the $y-$dimension in the main study cases).

Besides the validation purpose, the aim of this section is to present, from a practical point of view, the role of the model parameters. Therefore, we will present to the reader some keys to realistic fluid simulations. Athough all the introduction part on BGK models were discussed thouroughfully in the litterature, little exists on the subject addressed in this section, which is an original contribution of this thesis.

2.3.1 Boundary conditions

Since the emergence of simulation tools, either numerical or not, boundary conditions have always led the experimentalist to a dilemma: artificially setting *a priori* conditions to make the results look the way they are supposed to, or trying to find and model the exact laws of the phenomena. The solution resides in between and sometimes appears like a set of recipes (more or less justified). However, the correct recipe book will prevent the simulation from numerically blowing up or from emphasizing inappropriate behaviors.

In our modelling of a fluid through a tunnel (which is very similar to a laboratory wind tunnel), we must deal with several boundaries:

- initial conditions,
- solid-fluid interactions,
- input,
- output,
- boundaries parallel to the flow stream (upper and vertical ones, which shall model an open space with only a finite simulation domain).

Initial conditions

Initializing the system variables f_i's does not represent an unsuperable obstacle. However, some mistakes could generate early numerical instabilities and either make the simulation blow up or introduce spurious symmetries.

Therefore, the system is initialized as:

$$f_i(\mathbf{r},0) = 1 + \epsilon_0(\Xi(1) - 1/2) \tag{2.23}$$

where ϵ_0 is a small constant (order of 1%), and $\Xi(1)$ a random variable uniformly distributed over $[0,1]$, drawn for each $(i,\mathbf{r})$. The random component break some accidental symmetries (a numerical simulation of a flow around a cylinder will never generate Von Karman patterns if the variables are symetrical). But a small ϵ_0 prevents numerical instabilities from occuring during the first iteration (the smaller τ, the bigger the risk).

Last but not least, if C_{smago} is too low (in the case of highly turbulent fluid simulation, where instabilities are the most frequent), it may be a good idea to "warm up" the system: looping for several steps with a large τ $(= 1)$ before setting it to 0.5 and starting the subgrid process is often a solution that will prevent the simulation from blowing up.

Solid-fluid interactions: no-slip (bounce-back) condition

The underlying particle description of the fluid in the lattice techniques offers a simple and intellectually convincing way of solving this part of the problem. Indeed, the no-slip (or bounce-back) condition consist in reversing the velocity of particles entering a solidified site. If $\hat{i}$ labels the direction opposite to i (*i.e.* $c_{\hat{i}} = -\mathbf{c}_i$), it may simply be implemented as an exchange of densities $f_{\hat{i}} \leftrightarrow f_i$.

This method naturally leads to a zero-velocity by the solid level. More precisely, the position of the border of the modelled solid layer has been located half way between the solid cell and the neighboring free one [Hou *et al.* 1996].

Time dependent solid configuration

The aim of this thesis is to present a model of solid particle erosion/transport/deposition. Therefore, the shape of the solid configuration may evolve during the simulation.

However, with these simple rules, complex or time dependent solid configurations are not a problem anymore. Indeed, the bouncing back can be applied on a site at once and not at the following time step without any intrinsic model impossibility.

However, what happens to the $f_i(\mathbf{r}, t_1)$ if the site $\mathbf{r}$ becomes solid at time t_1 should also be clarified: in our opinion, a good solution is to compute at time t_1 a zero-velocity distribution ($\Rightarrow f_i(\mathbf{r},t_1) \leftarrow \rho(r,t_1)t_{\|\mathbf{c}_i\|}$) and trap the fluid into the solidified cell until it is released.

During the following time steps, the fluid evolution algorithm is not performed until the site is unsolidified, if ever. There are two advantages of choosing this option:

1. the total fluid density is conserved;

2. when a site is unsoidified, it is already set in a natural (equilibrium) zero velocity distribution, preventing from a blowing up risk, contrary to, for example, if all the local fluid density had been stored in the "rest" componenent).

Porous solid

The lattice resolution may be not precise enough to exactly reproduce an obstacle (*e.g.* a fence, built with spaced out planks) with a given porosity ϕ ($\phi = 0.0 \Rightarrow$ full obstacle, $\phi = 1.0 \Rightarrow$ no obstacle). But the previous idea can be slightly modified by bouncing the particle back only with a probability ϕ_f, *i.e.* replacing a fixed porosity by an averaged one. From a practical point of view, ϕ_f is obviously linked to ϕ and must be calibrated. Some results are proposed in section 2.3.7.

Input

In laboratory wind tunnel experiment, the input flow is forced by a jet. The idea is the same here, as the fluid entry velocity is set by the user on the left hand column of the computational domain.

Beware! The relaxation time on this boundary must therefore be 1 so the system reproduces the given entry speed exactly. Indeed, if the purpose if to set the velocity to u_0, the local fluid distribution must be forced to $f^{eq}(\rho, \mathbf{u}_0)$.

Moreover, one should let some horizontal distance (roughly one time the height of the domain) before the correct velocity profile is established.

The solution of accelerating the fluid only from the top boundary, has been tested but abandoned as the acceleration was not reaching the lowest layer (or providing very poor velocity profiles).

Accelerating the flow by adding momentum on random sites was also abandoned, except in the specific case of the BGK-multiparticle model for which satisfactory results for the Poiseuille flow experiment were obtained (cf. appendix A).

Ouput

A simple solution, which has been adopted, is to apply exactly the same method as for the input condition as exposed in the previous paragraph.

Some zero-gradient methods have been tried with some success but were necessary only with the early Lattice Bolzmann directly inspired from the cellular automata collision rule (section 2.1.2). The best one consists in copying the distribution from a vertical layer 20 sites away from the end of the tunnel to the last vertical layer [Masselot 1995].

Input/output boundaries: a precision about cyclic conditions

On the one hand, when fluid condition are cyclic, one must verify that the solid configuration is also cyclic. On the other hand, when forcing boundary condition, it is recommended to set locally the relaxation time to $\tau = 1$, so that the velocity will be forced exactly to its given value.

Upper boundary

When the fluid is not accelerated from above, the aim is to model an infinite boundary condition at the "sky" level. A classical solution with other numerical approaches is to compute larger and larger cells in the upper part of the system; however, in lattice models, the cell size is uniform. Moreover, to get shorter computation times, one wishes to get domain size as small as possible and is therefore driven to imagine alternatives to the brute force solution consisting in piling up cell rows.

A zero-gradient method has also been tested, trying to copy the second highest raw onto the highest one. The main drawback was to accentuate tiny fluctuations in a non-natural way.

The chosen solution only consists in applying the zero-gradient method but setting the vertical component of the fluid velocity to 0 (τ must therefore also be forced to be 1 there).

For the bottom boundary, in case of absence of solid (in the same flow experiment as in figure 2.9 for example), the same method is applied.

Initial coditions: lowering the blowing up risk

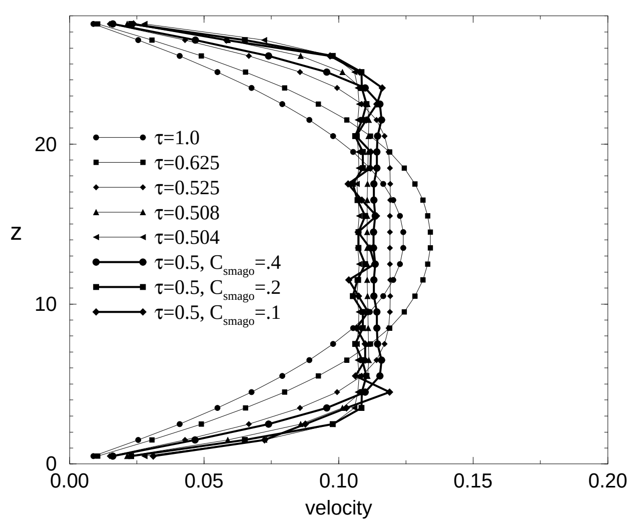
Starting a simulation directly in low viscosity mode (*i.e.* $\tau = 0.5$, C_{smago} low) can make the simulation blow up in the very first steps. This phenomenon is emphasized when the solid landscape is uneven. It may be useful in such situations to make the simulation run for several time steps (*e.g.* twice the highest domain dimension) with "calmer" parameter ($\tau = 1$) before tuning them to reach the highest Reynolds numbers possible.

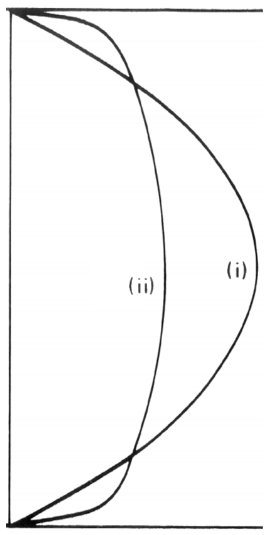
2.3.2 Pipe flow

The pipe flow problem may be considered as a fluid moving through two horizontal planes, without influencing borders in the $y-$direction.

The purpose of this first experiment is to evaluate the influence of τ and C_{smago}. Figure 2.5 describes how the flow can change from laminar to turbulent, comparing different velocity profiles with a reference [Tritton 1988, p.338-339].

Moreover, the role of the subgrid extrapolation is demonstrated in figure 2.6: the averaged velocity profile is the same with a subgrid model as without (with a





a) *b)*

Figure 2.5: The pipe flow experiment, where the averaged velocity profile is plotted for different set of relaxation times and Smagorinski constants in *a)*. Compared to the qualitive separation laminar *(i)*/turbulent *(ii)* by [Tritton 1988] in *b)*, this experiment shows how the value of the two key parameters influence the resulting simulation. Domain size $n_x \times n_y \times n_z = 120 \times 3 \times 30$.

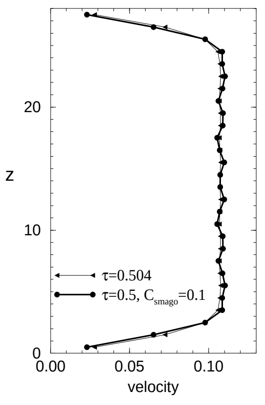
small relaxation time); however, looking closer at a simulation snapshot reveals the complex structure of the flow in the subgrid model simulation.

With this experiment, with have clearly shown how the theoritical link betwen the relaxation time and the viscosity (cf. equation (2.10)) can be pointed out. Moreover, once a turbulent profile is reached, the subgrid model can more or less exhibit small eddy structures.

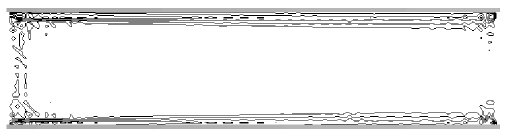
2.3.3 Open channel

The open channel experiment is simply a fluid flow over a flat ground, with no upper influencing boundaries; it corresponds to the situation of the wind blowing over a flat surface. The boundary conditions are set according to those of section 2.3.1: entry velocity is set to 0.1 on the left and right hand planes, zero-gradient with a zero z-component on the upper layer (the “sky”), and cyclic in the y-dimension.

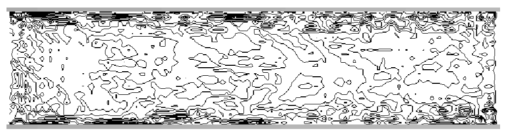
The same study such as in the previous experminent could be undertaken but would not be very interesting. However we may here focus on two characteristics



a)



b) $\tau = 0.504,\ C_{smago} = 0$



c) $\tau = 0.5,\ C_{smago} = 0.1$

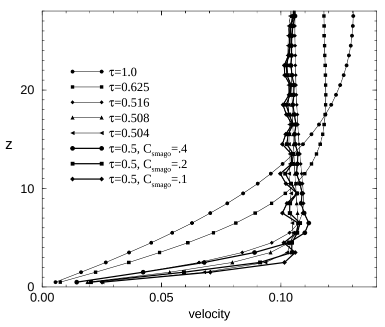
Figure 2.6: The same experiment as in figure 2.5 detailled for two sets of parameters, with and without the subgrid model activated ($C_{smago} > 0$). Although the averaged velocity profiles does not look very different in *a)*, a closer look at a flow snapshot of the vorticity isolines points out how the subgrid model in *c)* can exhibit complex structures of the fluid, totally ignored in *b)*.

of the flow:

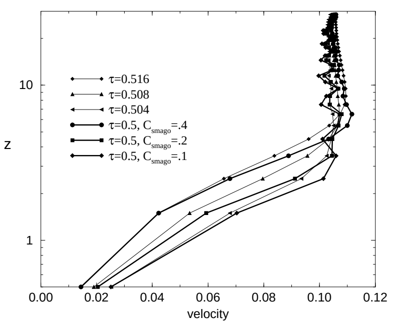
- the logarithm profile that should be recovered for turbulent flows (low enough relaxation time), according to [Tritton 1988]. These results are displayed in figure 2.7 *b)*;
- a closer look at the role of C_{smago}, considering the vorticity field structure near the ground for different values in figure 2.8.

From this experiment, we learn that turbulent conditions (*i.e.* when the velocity profile becomes logarithmic in its lower part) are reached with $\tau \leq 0.516$. However, the lower the relaxation, the more turbulent is the fluid (the maximum velocity is reached at a lower altitude).

Moreover, a too high C_{smago}, such as 0.4 in this case, has an effect opposite to that expected: instead of developping eddy structures, it only increases the viscosity. On the other hand, the lower is the Smagorinski constant, the more developped these structures are. However, if C_{smago} comes to be too small, the simulation can numerically blow up. Note that these threshold values for C_{smago} are valid for the simulations over a flat terrain, and are dependent on the solid landscape configuration.



a)



b)

Figure 2.7: Averaged velocity profiles for the open channel experiment. Figure *a)* shows the influence of the two key paramenters, in the same way as figure 2.5 *a)*. On *b)*, the same profiles are displayed (for a reduced set of parameters) with a logarithmic scale: the purpose of this features is to show how these parameters produce log curves in their lower part, according to the theory for turbulent flows by [Tritton 1988], where $u(z) = u_* \log \frac{z}{z_0}$, where u_* and z_0 are two parameters. Domain size $= 120 \times 3 \times 30$.

2.3.4 Flow around Cylinder

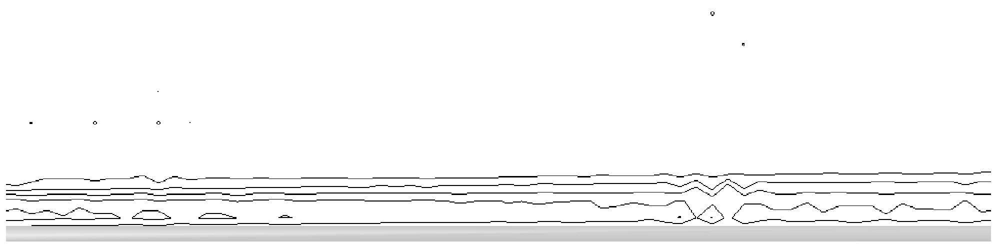
A classic experiment in the hydrodynamic litterature is the fluid flow perpendicular to a cylinder. Despite its simplicity, this study offers a look to a large panel of fluid behaviors when modifying the Reynolds number.

This exploration is usually done by increasing the entry velocity speed in a wind tunnel; it can also be achieved with a numerical model by lowering the viscosity (*i.e.* lowering the relaxation time and the Smagorinski constant). From a practical point of view, we will focus on the flow shape in figure 2.9.

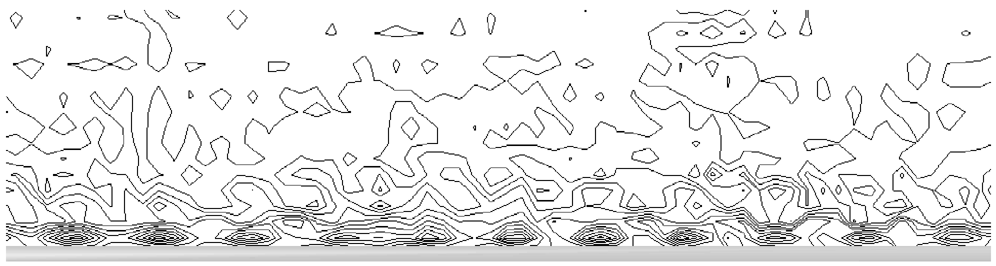
Results show the classical evolution between a symetric laminar flow and a more and more disturbed Von Karman vortex street, in the same manner as in [Tritton 1988].

2.3.5 Trenches: low Reynolds number experiments

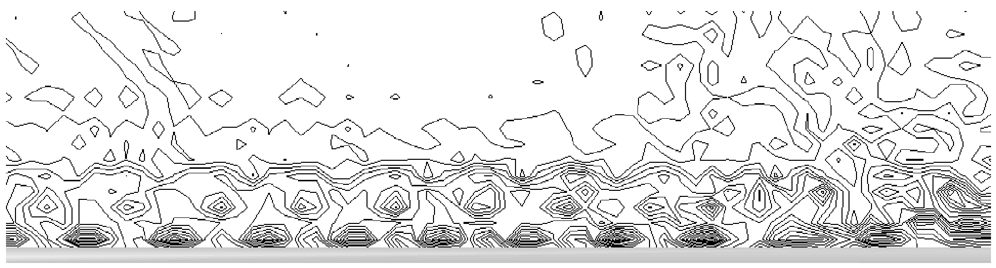
Although reaching high Reynolds number flow remains the main motivation of this numerical fluid model, it may also be used to simulate lower one. The comparison has been held between numerical results and wind tunnel experiments for flows over various shape of trenches, for which streamlines are displayed in figure 2.10. Numerical experiments agree perfectly with indoor wind tunnel ones by [Taneda 1979].



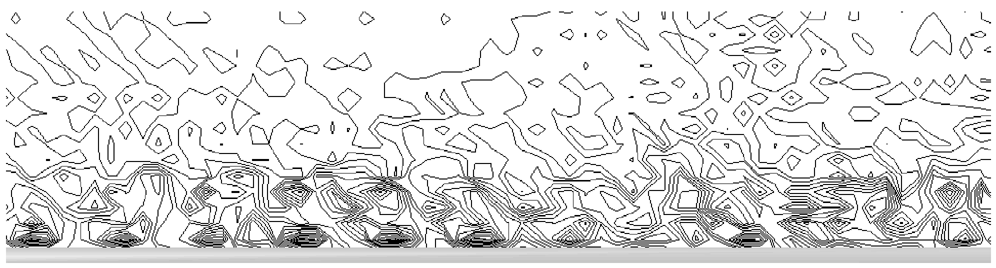
$C_{smago} = 0.2$



$C_{smago} = 0.1$



$C_{smago} = 0.09$



$C_{smago} = 0.085$

Figure 2.8: In the open channel experiment, vorticity isolines for different C_{smago} ($\tau = 0.5$). The averaged velocity profiles for these parameters do not sensibly vary one from each others. However, a closer look at the ground level shows how the smaller C_{smago}, the more developped the eddies.

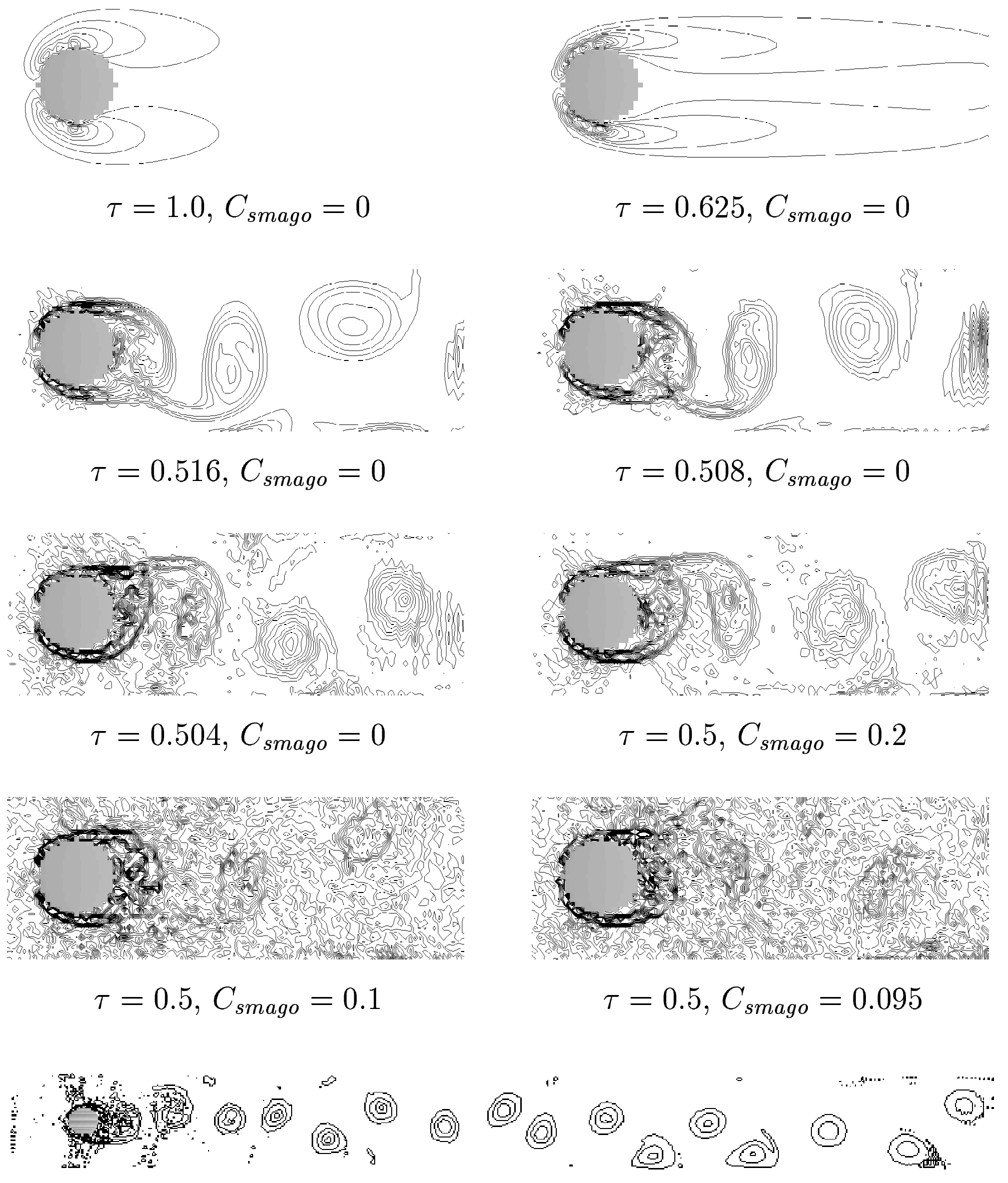


Figure 2.9: Vorticity isolines of fluid flows around a cylinder from our numerical simulations, with different viscosity (tuned lowering τ and seting C_{smago}). C_{smago} lower than 0.09 inevitably drive the simulation into a numerical blowing up.The lowest figure shows the same experiment for a much longer tunnel, and the long range development of a Von Karman vortex street [Taneda 1979, Sumer *et* Fredsøe 1997]. Domain size for the eight upper figures $150 \times 3 \times 40$; for the lowest one $500 \times 3 \times 50$.

Streamlines or streaklines?

The mix between both representations of fluid velocity fields has often been done. The opportunity is given here to make the distinction.

Both are the representation of the trajectories of a particles following the fluid flow. On the one hand, for *streamlines*, these particles are moving over a "spapshot" fluid velocity field (the numerical simulation is stopped, then the streamline computed, for an easier post-experimental rendering); on the other hand, for *streaklines*, the particles are released and are driven by the time-evolving fluid flow (this can more easily be achieved in wind tunnel experiment, where particle can for example be smoke released at different tunnel locations and drifted by the fluid).

For laminar flows, both streamlines and streaklines will give the same results as the flow is stationary. But for non-steady flows, the visual results will not be the same.

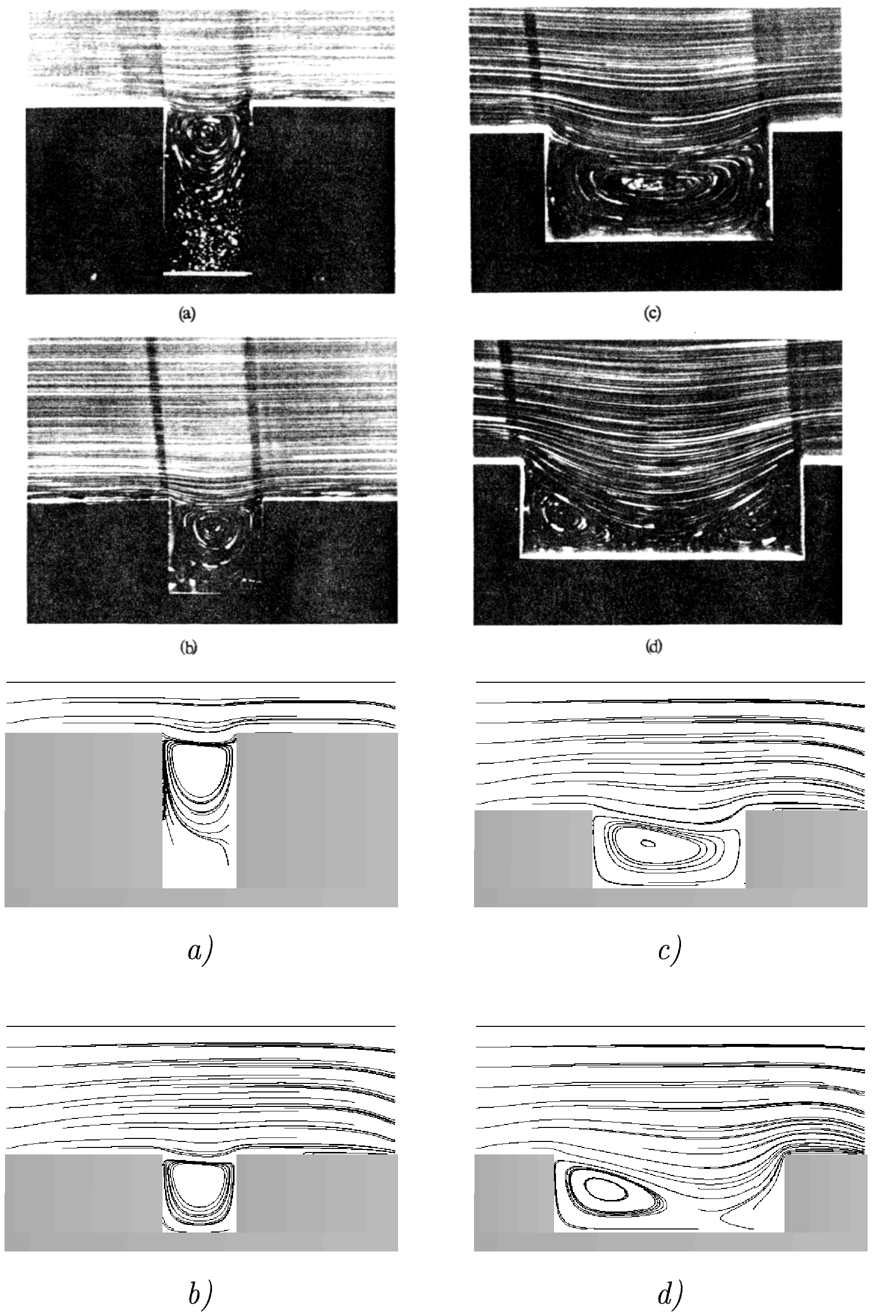


Figure 2.10: Fluid streamlines over trenches of different shapes; comparison between an indoor wind tunnel at Reynold number = 1.2×10^{-2} by [Taneda 1979] (upper results) and numerical output of our model ($\tau = 1$). For every trench, h is the height, and l the length; *a)* $l/h = 0.5$; *b)* $l/h = 1$; *c)* $l/h = 2$; *d)* $l/h = 3$. Domain size = $150 \times 3 \times 70$.

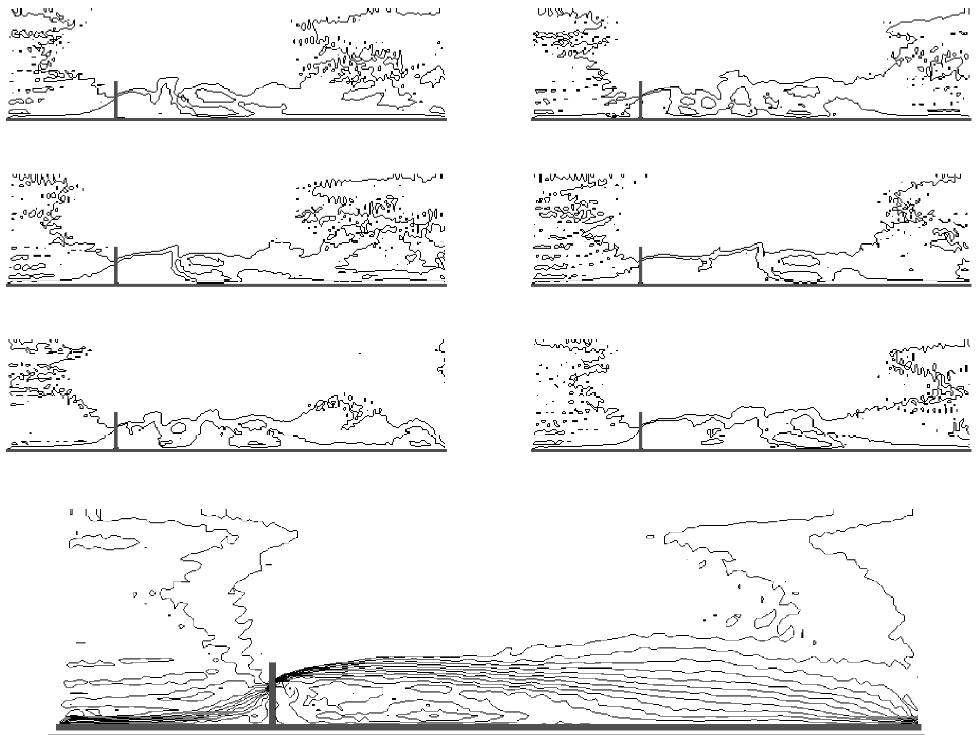


Figure 2.11: Velocity isolines around a full fence as obtained in simulations. The six upper figures are snapshots, every 1000 time steps, with very different instantaneous flow configurations. The lower one shows the averaged flow over 50 samples (every 1000 steps).

These last three experiments are more dedicated to the motivation of the current work: modelling a flow similar to a wind strong enough to transport snow particle around a structure.

2.3.6 Flow around a fence and hill

As this situation is often found in reality, we have faced our model to indoor experiment of flow around a fence in front of hills of different shapes by [Castelle *et al.* 1992].

The results presented in this reference work are averaged measurments of the flow field, but our model can produce quite developed turbulent flows (see figure 2.11). We have therefore averaged numerical output over several time steps to obtain a mean flow representation, which coincide very well with the cited work, in term of fluid velocity isolines (figure 2.12).

The flow structure around fences windwards slopes with different steepness is well recovered and will allow us to later study the particles transport and deposition around such structures.

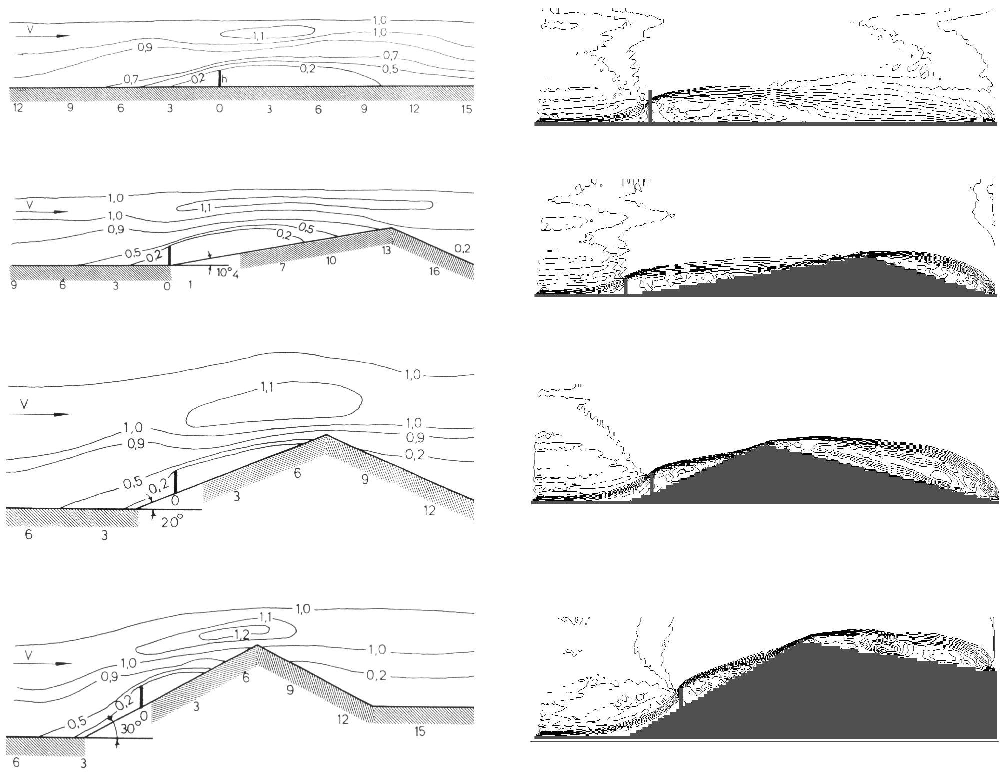


Figure 2.12: Velocity isolines around a full fence and a hill. Our numerical averaged results (cf. figure2.11) are compared with indoor wind tunnel experiments by [Castelle *et al.* 1992]. In the experimental results, ony some isolines (relatively to the entry velocity) are plotted; in the numerical one, isolines are plotted every 10% of the entry velocity. The comparison between both results, focusing on the fluid flow around the ground in the area ranging from the fence to the top of the hill is good, as the influence of the obstacle is correctly caught on these various configurations. $\tau = 0.5$, $C_{smago} = 0.15$, domain size=$150 \times 3 \times 60$.

2.3.7 Porous fences and bottom gaps

For outdoor works, it has been shown that efficiency of fence snow storage is increased by adding a ground clearance under the fence or using porous fences, in order to decrease the turbulence in the low velocity area leeward the fence. We therefore have tested these features with several parameters and compared them with litterature results:

- **Bottom gaps:** in figure 2.13, numerical simulations outputs are displayed for various bottom gap sizes. The average flow measurements show how the velocity is lowered leeward the obstacles, meanwhile flow snapshots reveals how the eddy activity is decreased at the ground level (this will be important to lower the erosion leewards the fence, thus to increase the snow storage efficiency).

- **Porous fences:** as indicated in section 2.3.1, a porous site can be modelled as a site on which particles bounce back only with probability ϕ_f. In the same manner as for the bottom gap fences, averaged and instant velocity fields are displayed in figure 2.14 for different porosities; moreover, they can be qualitatively compared to results from litterature in figure 2.15. However, the probability ϕ_f is not directly the real porosity modelled and some calibrations have been undertaken comparing horizontal and vertical velocity profiles with indoor wind tunnel experiments in figures 2.16 and 2.17).

For both kind of works, the numerical model catches the effect of the modifications from the full fence. However, in outdoor constructions, both setting a ground clearance or porosity involve small scale modifications: as they are too small to be put as-is directly in a simulation (due to the lattice cell resolution), some calibrations have had to be undertaken.

2.4 Summary

The purpose of this chapter was to present the fluid model used in this work, from several aspects:

1. Introducing the model, its evolution from the first stages up to recent developments. There is no original research there, as every features can be found in literature or even in complete state-of-the-art review articles.

2. However, considering the lattice domain as a virtual wind tunnel, original solutions had to be developed, mainly for boundary conditions. Moreover, the parameter space had to be explored to recover situations found in the literature.

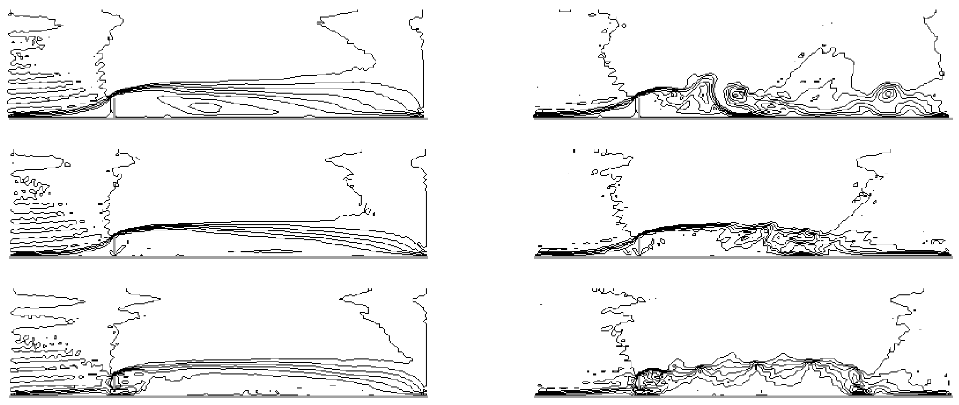


Figure 2.13: bottom gap fence experiments. For different ground clearance (1/9, 2/9 and 3/9 from top to bottom). On the left hand column, the averaged velocity field has been measured, while on the right hand side, a typical velocity isolines snapshots is displayed for each gap height. We can observe on the snapshots how turbulence at the ground level is lowered when the gap is larger, increasing the efficiency of the work in term of snow storage.

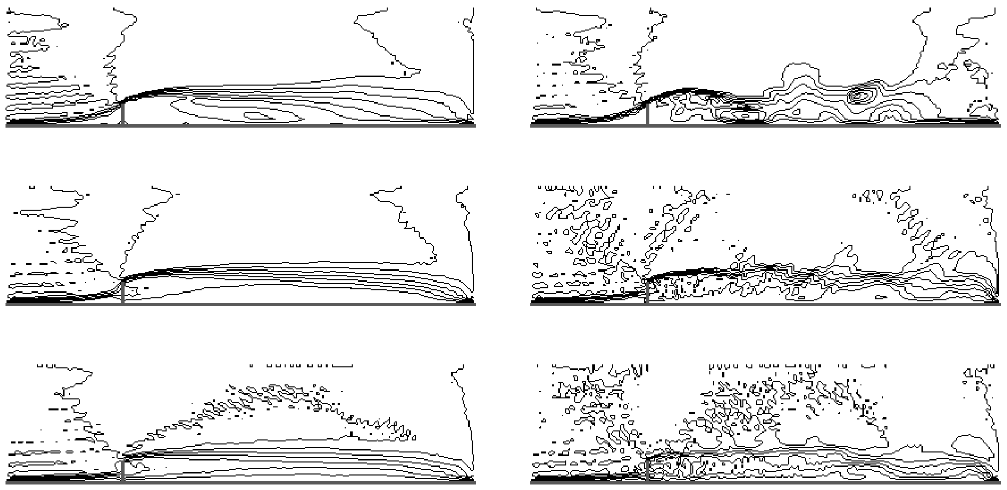


Figure 2.14: experiments with fences of different porosities ($\phi_f = 0., 0.9, 0.95$). Isolines are plotted (snapshots on the right hand side and averaged results on the left hand one) and can be compared with those of figure 2.15, showing how the average leewards eddy is avoided, and the velocity smoothly reduced at the ground level.

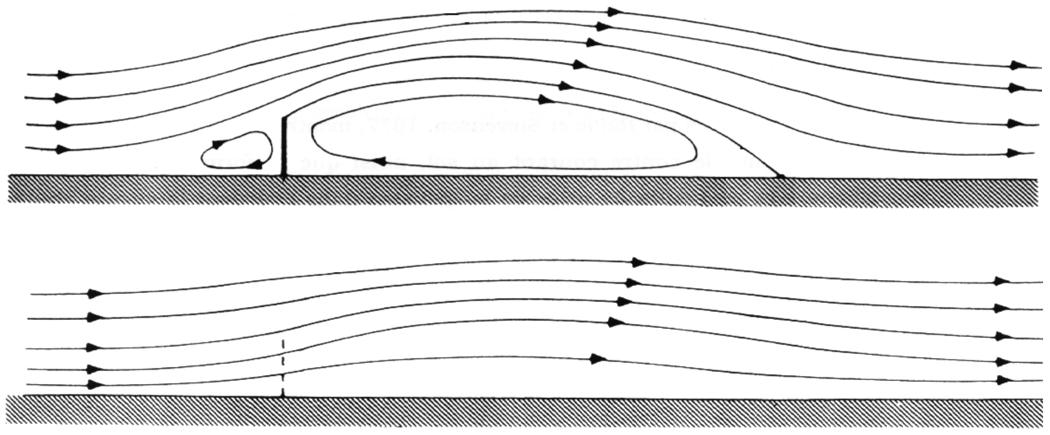


Figure 2.15: fluid streamlines on a full fence and on a $\phi = 50\%$ fence by [Raine *et* Stevenson 1977].

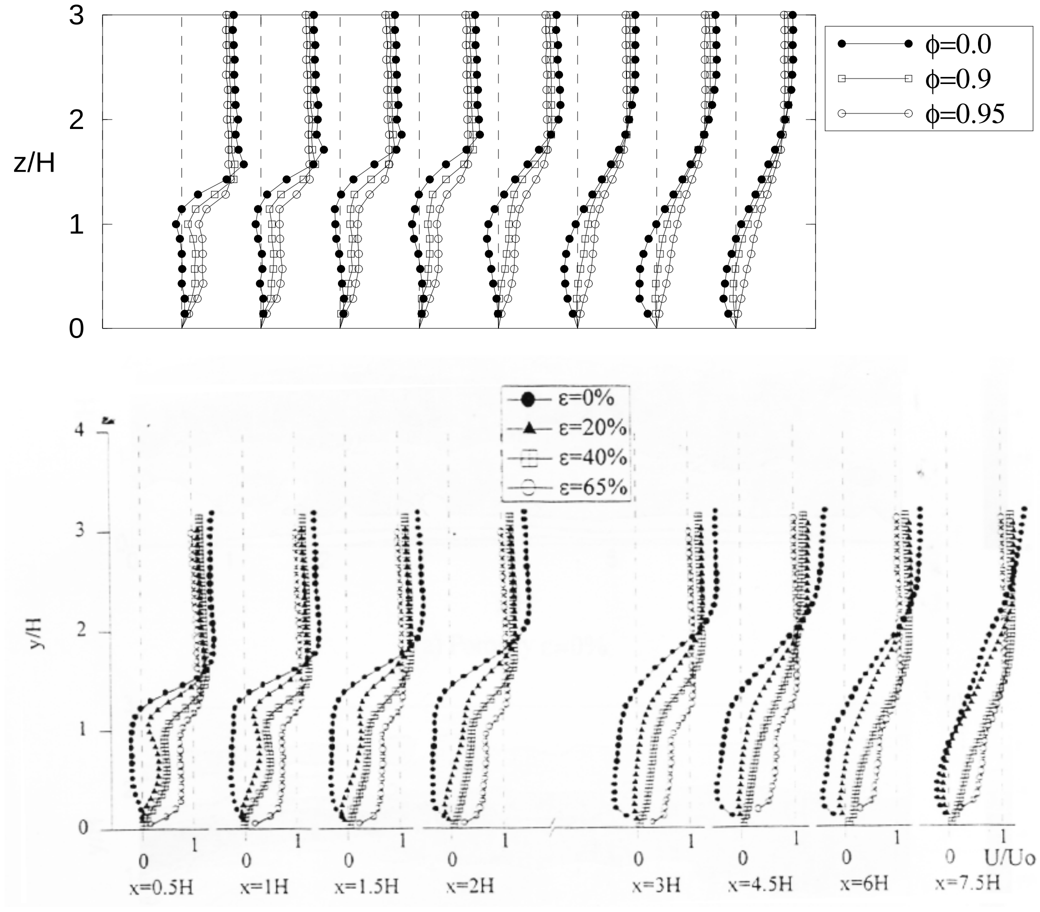


Figure 2.16: horizontal velocity profiles are measured at various locations, leewards fences (at positions $x = 0.5H$, $x = H$, $x = 1.5H$ ... where H is the fence height). Experiments have been achieved for four different porosities. Numerical experiments results are displayed on the upper part of the figure (numerical porosity is represented by ϕ and the height in respect to the fence height by z/H) and wind tunnel results from [Lee 1998] on the lower part (porosity is ϵ, and the height y/H, and the Reynolds number is 10^4).

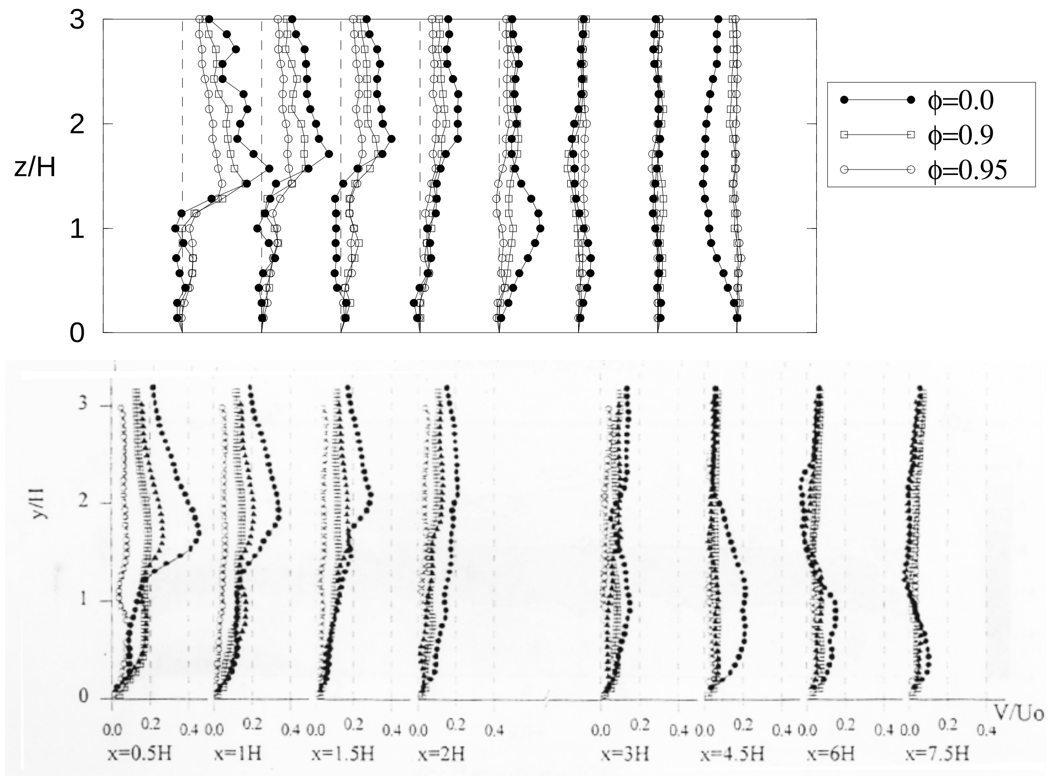


Figure 2.17: in the very same manner as in figure 2.16, vertical velocity profiles are plotted for both numerical and wind tunnel experiments.

3. Numerical experiments have been ran, focusing on turbulent flows in a wide range of situations, from theorical cases to flow around outdoor structures.

As a conclusion, we have shown in this chapter how lattice models are well suited to model turbulent flows in various situations and can be considered as an alternative to classical models. Moreover, as they are based on regular grid, local computations, they easily handle time dependent solid configuration. This main feature will be a key point in the modelling of solid particle transport, deposition and erosion.

Chapter 3

Solid Particles Model

3.1 Basics mechanisms

Classical approaches to model solid particles behavior under the action of a fluid field can historically be split among two methods, roughly:

- The Eulerian one, where, on each cell of a domain grid, density and other parameters of the solid phase are averaged. The model is therefore governed by equations stating (explicitly or implicitly) the exchanges between adjacent cells.
- The Lagrangian method, modeling individually a set of particles (and tracking their exact positions, rotation speeds …). These solid particles interact with the fluid (which may itself be modeled through an Eulerian approach) and between each other with mechanisms similar to molecular dynamics. This later method better suits problems with smaller amounts of particles, and offers greater opportunities to directly model physical laws concerning the individual particles.

From this first point of view, the cellular automata approach for the solid phase lies somewhere between both, as on the one hand, the domain is decomposed in cells and there is no information of exactly where, inside a cell, a particle should be. On the other hand, particles are indivisible and can be individually tracked in the Lagrangian way.

3.1.1 Notations

With the same approach as in section 2.1.1, we represent the solid particles as quantities evolving synchronously and discretly over the same lattice Γ. At time t, on site $\mathbf{r}$, the number of particles traveling with velocity $\mathbf{c}_i$ is noted $p_i(\mathbf{r}, t) \in \mathbb{N}$, and $p_0(\mathbf{r}, t)$ is the number of particles remaining on the site. Contrary to the lattice Boltzmann approach (see section 2.1.2), p_i are integer values, therefore the exclusion principle is not taken into account (several particles can be streamed along a single lattice link) and the particles are represented as discrete variables.

Thus, we can define the local particle density:

$$\rho_p(\mathbf{r}, t) = \sum_{i=0}^{q} p_i(\mathbf{r}, t) \tag{3.1}$$

And the particle flux:

$$\mathbf{J}_p(\mathbf{r}, t) = \sum_{i=1}^{q} p_i(\mathbf{r}, t)\mathbf{c}_i \tag{3.2}$$

where q is the number of lattice cell neighbors. This number can be different than q_f, the number of links in the fluid model: for example, a three dimensional fluid model $D3Q19$ needs only the 18 neigbors at distance 1 and $\sqrt{2}$ when a cell could also be linked to neighbors at distance $\sqrt{3}$, letting $q = 26$.

From an implementation point of view, the evolution from $p_i(\mathbf{r}, t)$ to $p_i(\mathbf{r} + \mathbf{c}_i, t + 1)$ would be too heavy to express it on one stage. We may therefore use a temporary variable $p^{(i)}(\mathbf{r}, t)$, collecting all the contributions throughout the different steps of the particle distribution evolution algorithm, for particles from $\mathbf{r}$ aiming to be streamed towards $\mathbf{r} + \mathbf{c}_i$.

For these particles, we may now define transport mechanisms, *i.e.* models for aeolian motion, deposition and erosion, as well as stating domain boundary and input conditions.

3.1.2 Particle motion

On the site $\mathbf{r}$, at time t, a particle may follow the fluid velocity $\mathbf{u}_f(\mathbf{r}, t)$ as defined in equation 2.4 plus a falling velocity (taking into account gravity force) $\mathbf{u}_{fall}$ to fulfill:

$$\mathbf{J}_p(\mathbf{r}, t) = \rho_p(\mathbf{r}, t)\left(\mathbf{u}_f(\mathbf{r}, t) + \mathbf{u}_{fall}\right), \tag{3.3}$$

where the parameter $\mathbf{u}_{fall}$ can be calibrated through its ratio versus $\mathbf{u}_f^{\infty}$ (this ratio can be easily extracted from field experiments).

Following exactly such a law would violate the lattice principle stating that particle must remains on the lattice nodes. The solutions presented here are probabilistic methods that will satisfy (3.3) on average and minimize the diffusion.

$D = 1$ case

The one-dimensional case is rather simple. As shown in figure 3.1, the ideal position for a particle would be $\mathbf{r} + (\mathbf{u}_f + \mathbf{u}_{fall})$ (we may define $\xi_x = |\mathbf{u}_f + \mathbf{u}_{fall}|$). As the position is off lattice, we propose a probabilistic algorithm:

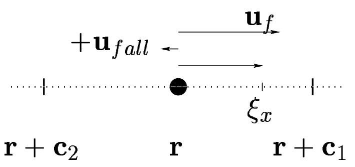


Figure 3.1: over a one dimensional lattice, a particle should follow the fluid velocity $\mathbf{u}_f$ plus an external velocity $\mathbf{u}_{fall}$. Therefore, it jumps from $\mathbf{r}$ to one of its neighbors ($\mathbf{r}+\mathbf{c}_1$ or $\mathbf{r}+\mathbf{c}_2$, depending on which one $\mathbf{u}_f+\mathbf{u}_{fall}$ points towards) with probability $\xi_x = |\mathbf{u}_f + \mathbf{u}_{fall}|$ and rest on $\mathbf{r}$ with probability $(1-\xi_x)$.

- the direction of motion depends on the sign of $\mathbf{u}_f + \mathbf{u}_{fall}$
- the particle moves with probability ξ_x, and rests with probability $(1-\xi_x)$

This algorithm can be staightforwardly implemented, and the mean position will trivially be the right one. We will see later how to manage efficiently the situation where many particles are to be dispatched from the same site.

$D = 2$ case

The technique used is about the same as the previous one, based on a probabilistic spread of the particle from $\mathbf{r}$ to its neighbors. The first stage is to define which quadrant the particle is leading towards (north-east, north-west, south-west or south-east); this first decision is only based on the sign of the $x-$ and $y-$components of $\mathbf{u}_f + \mathbf{u}_{fall}$.

Then, the probabilities of the horizontal and vertical motions can be defined as $\xi_x = |\mathbf{u}_{f\,\mathbf{x}} + \mathbf{u}_{fall\,\mathbf{x}}|$ and $\xi_y = |\mathbf{u}_{f\,\mathbf{y}} + \mathbf{u}_{fall\,\mathbf{y}}|$ (where $\mathbf{v}_\mathbf{x}$ is the x-component of vector $\mathbf{v}$). Therefore, the probabilities of resting or reaching each of the three neighbors in the designated quadrant are (see figure 3.2):

$$\begin{array}{ll} \text{moving along the diagonal} & \xi_x\xi_y \\ \text{moving to the x-neighbor} & \xi_x(1-\xi_y) \\ \text{moving to the y-neighbor} & \xi_y(1-\xi_x) \\ \text{resting} & (1-\xi_x)(1-\xi_y) \end{array} \tag{3.4}$$

Trajectory error estimation

A deviation from the exact Lagrangian path of the particle is intrinsic to such an approach. For simplification purpose, we will suppose here that the vector $\mathbf{u}_f + \mathbf{u}_{fall}$ points north-eastwards. We shall now focus on the probabilistic behavior of the algorithm.

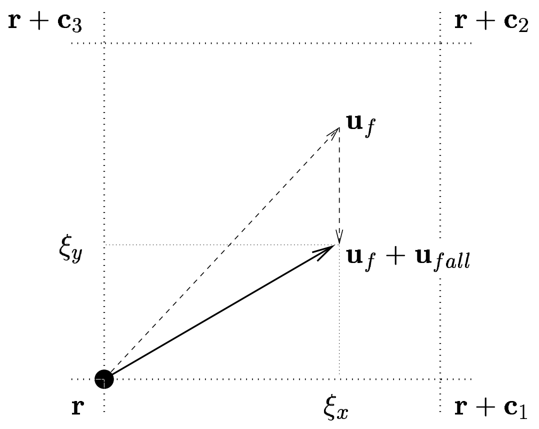

$\mathbf{r}+\mathbf{c}_3$
$\mathbf{r}+\mathbf{c}_2$
$\mathbf{u}_f$
ξ_y
$\mathbf{u}_f+\mathbf{u}_{fall}$
$\mathbf{r}$
ξ_x
$\mathbf{r}+\mathbf{c}_1$

Figure 3.2: on a two-dimensional square lattice, once the right quadrant has been defined (in this particular case, the north-east one), the particle can be randomly spread among four possible neighbors. The probabilities for these four destinations are combinations of ξ_x and ξ_y, where the probability of moving to the site $\mathbf{r}+\mathbf{c}_2$ is $\omega_2=\xi_x\xi_y$, to $\mathbf{r}+c_{1,3}$, $\omega_{1,3}=\xi_{x,y}(1-\xi_{y,x})$, and the probability to remain on site $\mathbf{r}$ is $\omega_0=(1-\xi_x)(1-\xi_y)$.

The averaged position is the right one, as:

$$\underbrace{\xi_x(1-\xi_y)}_{\omega_1}\begin{pmatrix}1\\0\end{pmatrix}+\underbrace{\xi_x\xi_y}_{\omega_2}\begin{pmatrix}1\\1\end{pmatrix}+\underbrace{\xi_y(1-\xi_x)}_{\omega_3}\begin{pmatrix}0\\1\end{pmatrix}+\underbrace{(1-\xi_x)(1-\xi_y)}_{\omega_0}\begin{pmatrix}0\\0\end{pmatrix}=\begin{pmatrix}\xi_x\\\xi_y\end{pmatrix}$$

The mean Euclidean deviation is:

$$\sigma_p(\xi_x,\xi_y)=\sum_{i=0}^{3}\omega_i\left\|\mathbf{c}_i-\begin{pmatrix}\xi_x\\\xi_y\end{pmatrix}\right\| \tag{3.5}$$

Its maximum is realized when $\xi_x=\xi_y=1/2$. Averaging over ξ_x and ξ_y, as the transport algorithm is iterated n times. Thus, $\sigma_p^{(n)}$ varies as or a random walk (n independent trials) around the average position:

$$\sqrt{n}\int_0^1\int_0^1\sigma_p(\xi_x,\xi_y)d\xi_x d\xi_y\approx 0.521\sqrt{n} \tag{3.6}$$

and the integral term is the *mean diffusion coefficient.*

The triangular approach

Instead of spreading a particle from $\mathbf{r}$ among four sites, another technique can also be imagined. Rather than determining the quadrant in which the particle should go, we can restrain it to a triangle. In the case proposed in figure 3.2, the

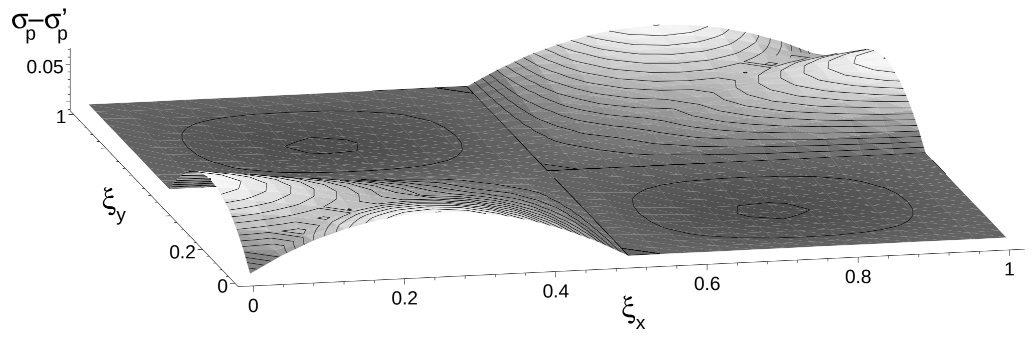


Figure 3.3: to compare the three- and four-neighbor spreading algorithms, $\sigma_p(\xi_x,\xi_y)-\sigma'_p(\xi_x,\xi_y)$, from equations (3.6) and (3.8) is displayed here with its isolines. Where this difference is positive, *i.e.* $(\xi_x,\xi_y)\in[0,\frac{1}{2}]\times[0,\frac{1}{2}]\bigcup[\frac{1}{2},1]\times[\frac{1}{2},1]$, the later one is the best (as the mean Euclidean error is lower).

particle should only be spread between sites $\mathbf{r}$, $\mathbf{r}+\mathbf{c}_1$ and $\mathbf{r}+\mathbf{c}_2$. The probability associated with these three positions, namely ω'_0, ω'_1 and ω'_2, should therefore be defined to fulfill $\omega'_1\mathbf{c}_1+\omega'_2\mathbf{c}_2=(\mathbf{u}_f+\mathbf{u}_{fall})$ and $\omega'_0+\omega'_1+\omega'_2=1$ (the ideal position is therefore the geometrical center of mass of the weighted triangle):

$$\omega'_0=1-\xi_x,\qquad \omega'_1=\xi_x-\xi_y,\qquad \omega'_2=1-\xi_x \tag{3.7}$$

The averaged position is the expected one, and the mean Euclidean deviation is

$$\sigma'_p(\xi_x,\xi_y)=\sum_{i=0}^{2}\omega'_i\left\|\mathbf{c}_i-\begin{pmatrix}\xi_x\\ \xi_y\end{pmatrix}\right\| \tag{3.8}$$

It is found that the mean Euclidean deviation (or the mean diffusion coefficient), in the sense of (3.6), is approximatively 0.518.

The best approach?

The finest approach is the one which minimize $\sum_i\omega_i\left\|\mathbf{c}_i-\binom{\xi_x}{\xi_y}\right\|$. At the first glance the second one appears to be the best, as a particle has less opportunities to diffuse away from its path, and the mean error is lower ($0.518<0.521$). But a closer look at the difference between σ_p and σ'_p for each (ξ_x,ξ_y), as displayed in

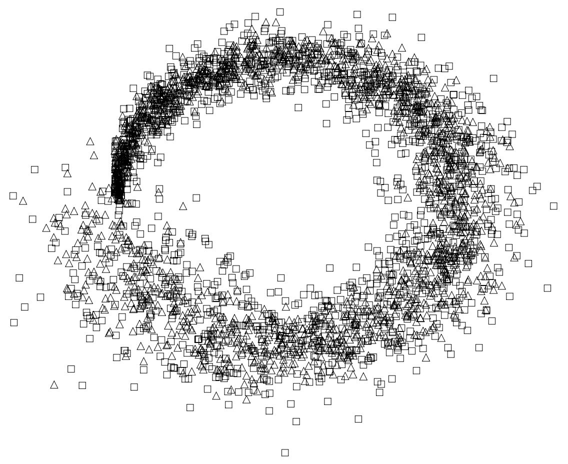


Figure 3.4: particle distributions when released from a point source and subject to a circular velocity field. The exact trajectory should therefore be a circle. The □ represent particles following the four-neighbor algorithm, and △ particles following the three-neighbors one. The figure shows, in agreement with theoretical predictions, that the first method diffuses slightly more than the second one.

figure 3.3, shows that the advantage of the first one is not true for every (ξ_x, ξ_y). Therefore, a better solution would be to choose the three-neighbors spreading method where $(\xi_x, \xi_y) \in [0, \frac{1}{2}] \times [0, \frac{1}{2}] \bigcup [\frac{1}{2}, 1] \times [\frac{1}{2}, 1]$, and the four-neighbor one elsewhere. Anyway, although the difference between both methods is sensible (as shown in figure 3.4), it is weak ($< 10\%$), and from a practical point of view, may be neglected.

A more general optimal method for the square lattice is possible: given (ξ_x, ξ_y), we must determine $\hat{\omega}_0$, $\hat{\omega}_1$, $\hat{\omega}_2$ and $\hat{\omega}_3$, optimizing the system:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} \hat{\omega}_0 \\ \hat{\omega}_1 \\ \hat{\omega}_2 \\ \hat{\omega}_3 \end{pmatrix} = \begin{pmatrix} 1 \\ \xi_x \\ \xi_y \end{pmatrix} \quad \begin{array}{l} \text{the sum of probability is equal to 1} \\ \text{the average } x\text{-position is } \xi_x \\ \text{the average } y\text{-position is } \xi_y \end{array} \tag{3.9}$$

$$\text{where} \quad \sum_{i=0}^{3} \hat{\omega}_i \left\| \mathbf{c}_i - \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix} \right\| \quad \text{is minimized}$$

Finally, it is interesting to note that the hexagonal lattice, where an approach similar to the triangle one can be applied, produces less diffusion as the mean Euclidean error, in the sense of (3.6), is as low as 0.25. But the hexagonal lattice offers less advantages from a practical point of view and cannot be extended to the third dimension.

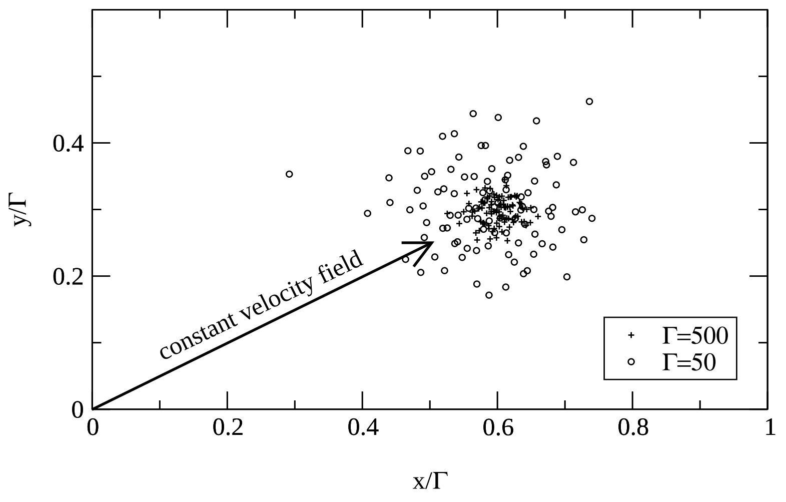


Figure 3.5: For two $2D$ square lattice of side size $\Gamma = 50$ and $\Gamma = 500$, particles are launched at the origin and submited during $1.2 \times \Gamma$ time steps to a velocity field equal to $(0.5, 0.25)$. The exact position is therefore $(.6\Gamma, .3\Gamma)$. This is the average computed position for both resolutions, however, for the finest resolution ($\Gamma = 500$), the standard deviation is smaller.

The lattice resolution

The lattice is a regular discretization of the modelled world. Therefore, the resolution of this discretizazion influences the precision of the computed trajectories.

As explained in equation 3.6, the mean Euclidian error for a trajectory of n steps varies as $\sqrt{n}$ times a standard error depending on the chosen distribution method. If the lattice resolution is refined by a factor γ, the same relative trajectory is covered in γn steps, thus the error is $\sqrt{\gamma n}$ times the same standard error. However, this Eucilidian error is defined versus the size of a cell, and the refined cell is γ times smaller. Therefore, the relative error varies as $\sqrt{\gamma n}/\gamma = \sqrt{n/\gamma}$.

In conclusion, we can say that refining the lattice resolution by a factor γ (thus multiplying by γ^d the number of lattice cells, thus by γ^{d+1} the number of time steps needed for the same trajectory computation) improves the precision by a factor $\sqrt{\gamma}$.

This improvement is displayed for a $2D$ case in figure 3.5. In this example, refining by a factor 10 the precision of the lattice for the same simulation results in getting 1000 times longer simulations and decreasing the standard deviation from 0.071 to 0.022 (*i.e.* a $3.23 \sim \sqrt{10}$ finer precision).

An “exact” method

A solution to drastically lower the trajectory error is to follow individually each particle, in a Lagrangian way, throughout the domain. We have tested this solution but, as it increases the computational load too much, it has been abandoned.

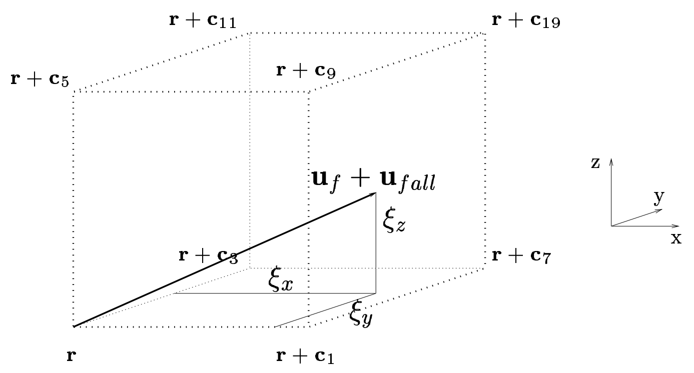

$\mathbf{r}+\mathbf{c}_{11}$
$\mathbf{r}+\mathbf{c}_{19}$
$\mathbf{r}+\mathbf{c}_{5}$
$\mathbf{r}+\mathbf{c}_{9}$
$\mathbf{u}_f+\mathbf{u}_{fall}$
ξ_z
$\mathbf{r}+\mathbf{c}_{3}$
$\mathbf{r}+\mathbf{c}_{7}$
ξ_x
ξ_y
$\mathbf{r}$
$\mathbf{r}+\mathbf{c}_{1}$
z
y
x

Figure 3.6: on a three dimensional lattice, once the right sector has been chosen, the particle can be randomly spread among 8 possible neihgors. The probabilities for these eight destinations are combinations of ξ_x, ξ_y and ξ_z.

$D = 3$ case

The method selected for the computations is a direct extension of the above square $D = 2$ method. The three canonical components of $\mathbf{u}_f + \mathbf{u}_{fall}$ give three probabilities ξ_x, ξ_y and ξ_z of moving among the $x-$, the $y-$ or the $z-$axis. We can therefore extract the probability:

$$\begin{aligned}
\omega_0 &= (1-\xi_x)(1-\xi_y)(1-\xi_z) \\
\omega_{xzy} &= \xi_x\xi_y\xi_z \\
\omega_{yz} &= \xi_y\xi_z - \omega_{xyz} \\
\omega_{xz} &= \xi_x\xi_z - \omega_{xyz} \\
\omega_{xy} &= \xi_x\xi_y - \omega_{xyz} \\
\omega_x &= \xi_x - \omega_{xy} - \omega_{xz} - \omega_{xyz} \\
\omega_y &= \xi_y - \omega_{xy} - \omega_{yz} - \omega_{xyz} \\
\omega_z &= \xi_z - \omega_{xz} - \omega_{yz} - \omega_{xyz}
\end{aligned} \tag{3.10}$$

where, by example, ω_{xz} is the probability of moving along the x and z directions, but not along the y one (the sign of the move depends only on the signs of the vector $\mathbf{u}_f + \mathbf{u}_{fall}$ respective components).

The distribution algorithm

Once the ω_i are known, we must build an algorithm to dispatch the particles from $\mathbf{r}$ among its neighbor $\mathbf{r} + \mathbf{c}_i$ with probability ω_i. If the number of particles $\rho_p(\mathbf{r}, t)$ is small, the first solution is, foreach particle, to randomly draw a direction according to the ω_i, as shown in algorithm 3.1.1.
Unfortunately, such a process can become tedious if $\rho_p(\mathbf{r}, t)$ is too large. It can therefore be less expensive, in term of computation time, to handle a binomial

distributed variable, namely $\mathcal{B}(n,p)$, representing the number of successful drawings within n trials, each of them with probability p of success. For large enough n ($\geq \theta_{\mathcal{B}} \approx 20$), $\mathcal{B}(n,p)$ can directly be extracted from a Gaussian distributed variable $\mathcal{N}(0,1)$ in one step [Knuth 1969]. Therefore, an improvement of the above algorithm can be proposed in 3.1.2.

Increasing the transport efficiency

If the fluid field $\mathbf{u}_f(\mathbf{r},t)$ was steady (*i.e.* $\mathbf{u}_f(\mathbf{r},t) = \mathbf{u}_f(\mathbf{r},t_0)$, $\forall t > t_0$), we could applied a time optimization to the above methods. For example, if the maximum fluid velocity $|\mathbf{u}_f(\mathbf{r})|$ over the $\mathbf{r}$ was $\mathbf{u}_{max} \leq 0.1$, we would be able to save a factor 10 in term of time steps for particle motion, when, instead of taking $\xi_{x/y/z} = (\mathbf{u}_{f\,\mathbf{x/y/z}} + \mathbf{u}_{fall\,\mathbf{x/y/z}})$, setting $\xi_{x/y/z} = 10\left(\mathbf{u}_{f\,\mathbf{x/y/z}} + \mathbf{u}_{fall\,\mathbf{x/y/z}}\right)$.

From a more general point of view, if we can extrapolate a maximum fluid velocity (depending on $\mathbf{u}_{entry}$, the landscape shape), we can therefore define a particle motion efficiency $\epsilon_p = 1/\max_{\mathbf{r},t}\mathbf{u}_f(\mathbf{r},t)$, and then take $\xi_{x/y/z} = \epsilon_p\left(\mathbf{u}_{f\,\mathbf{x/y/z}} + \mathbf{u}_{fall\,\mathbf{x/y/z}}\right)$. The result of this method is to uniformly (over the simulation domain and time) increase by a factor ϵ_p the moving particle flux, and to reach sooner the end of the simulation.

However, a problem arises if $\max_{\mathbf{r},t}\mathbf{u}_f(\mathbf{r},t)$ is under-estimated, as one of ξ_x, ξ_y, or ξ_z could once be greater than 1. In only this case, *i.e.* where $\xi_{max} = \max(\xi_x, \xi_y, \xi_z) > 1$, to avoid any probability greater than 1, but to keep the right direction of motion, we re-balance them: $\xi_x \leftarrow \xi_x/\xi_{max}$, $\xi_y \leftarrow \xi_y/\xi_{max}$, $\xi_z \leftarrow \xi_z/\xi_{max}$.

In this section, we have seen how to model aeolian transport of solid particles over a lattice, without splitting it among different phenomenon (creeping, saltation, suspension). However, we have not yet model deposition and erosion. Both algorithms presented in the folowing subsections seem rather simplistic, but they have shown to be enough to recover some interesting results.

3.1.3 Particle deposition

As defined in section 2.1.1, $s(\mathbf{r},t)$ is a boolean value indicating if the site $\mathbf{r}$, at time t, is solid. Therefore, if $s(\mathbf{r}+\mathbf{c}_i,t)$ is $true$, the particles directed from $\mathbf{r}$ in direction $\mathbf{c}_i$ cannot proceed as they would hit a wall. The solution is therefore to freeze them, and to increment consequently the local number of frozen particles $p_{frz}(\mathbf{r})$. Once the number of frozen particles exceeds a given threshold θ_{frz}, the site is solidified. This procedure is expressed in algorithm 3.1.3, and displayed in figure 3.7.

From a practical point a view, if θ_{frz} is too small, unrealistic deposit can build up (such as one site wide particle piles or arches). Meanwhile, setting a larger θ_{frz} ($\gtrsim 5$) averages the deposit and naturally avoids these model artifacts. The

Algorithm 3.1.1 a sequential particle distribution. $\mathcal{X}([0,1])$ is a random variable uniformly distributed in $[0,1]$.

```
do $i = 1, \rho_p(\mathbf{r},t)$
  $\pi \leftarrow \mathcal{X}([0,1])$
  $k \leftarrow 0$
  do while($\pi > \omega_k$)
    $\pi \leftarrow \pi - \omega_k$
    $k \leftarrow k + 1$
  end do
  $p^{(k)}(\mathbf{r},t) \leftarrow p^{(k)}(\mathbf{r},t) + 1$
end do
```

Algorithm 3.1.2 A faster particle re-direction process with binomial distributed drawings.

```
$\rho = \rho_p(\mathbf{r},t)$
$\xi \leftarrow 0$
do while ($\rho \geq \theta_{\mathcal{B}}$)
  !$\mathcal{B}$ extracted from $\mathcal{N}(0,1)$
  $\delta \leftarrow \mathcal{B}(\rho, \omega_\xi)$
  $p^{(\xi)}(\mathbf{r},t) \leftarrow \delta$
  $\rho \leftarrow \rho - \delta$
  !$\omega_i$ for remaining directions must sum to $1$
  $\bar{\omega} \leftarrow \sum_{i \geq \xi+1} \omega_i$
  do $i = \xi + 1, q$
    $\omega_i \leftarrow \omega_i / \bar{\omega}$
  end do
  $\xi \leftarrow \xi + 1$
end do

!not enough particle to extract $\mathcal{B}$ directly anymore
!the $\rho$ remaining particles must be redirected individually
do $i = 1, \rho$
  $\pi \leftarrow \mathcal{X}([0,1])$
  $k \leftarrow \xi$
  do while($\pi > \omega_k$)
    $\pi \leftarrow \pi - \omega_k$
    $k \leftarrow k + 1$
  end do
  $p^{(k)}(\mathbf{r},t) \leftarrow p^{(k)}(\mathbf{r},t) + 1$
end do
```

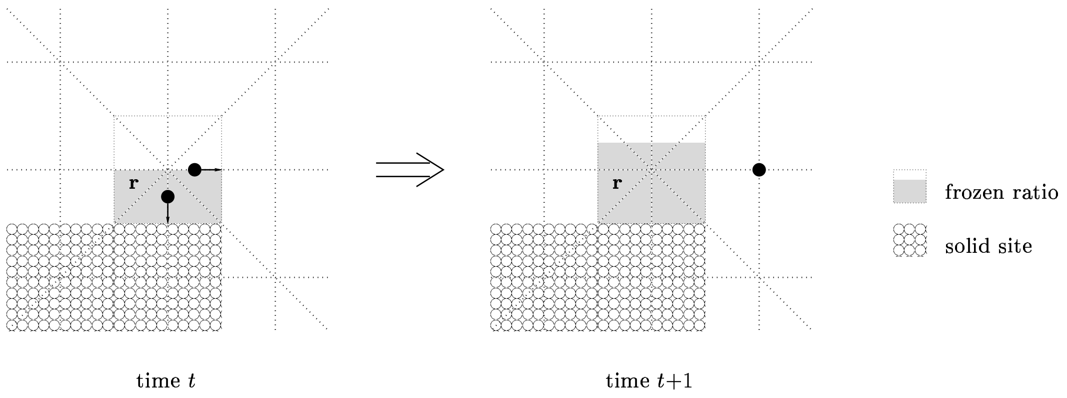


Figure 3.7: the deposition process, when two particles are leaving site **r**. The one directed rightwards can proceed normally as its destination is free. Meanwhile, the one directed downwards is heading towards a solid site; therefore, it will have to freeze on **r**, and increase the local frozen number of particles. Once this number reaches θ_{frz}, the site **r** is solidified.

parameter θ_{frz} is therefore a key to calibrate the "real" cell size (roughly, the more frozen particles are needed to solidify the cell, the wider it is), and ranges between 10 and 10^4 in our simulations.

A toppling process can be incorporated in this model (as it has been, in a simple version, with the 3D model). Such a process can take into account critical slope angles, such as described in [Dupuis *et* Chopard 2000a].

3.1.4 Erosion

Many features can be incorporated in the erosion process: wind velocity profile characteristics [Sundsbo 1997], particle concentration in the fluid calming down vortices thus the erosion rate [Castelle 1995], landing particles shocking the deposit and ejecting (instantaneously or after a short delay) other particles [Martinez 1996].

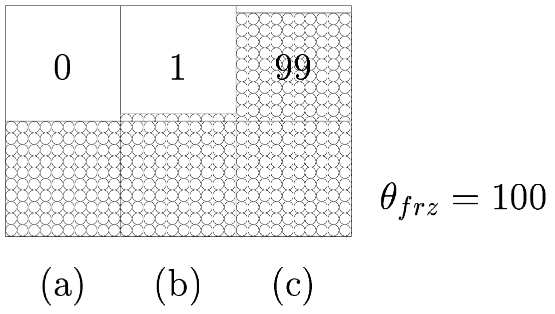
We did try some of the above ideas to recover this complex phenomenon, but the method finally kept is a large simplification of them. The idea is simply, to release frozen particles (see previous section) with a probability ζ_p, and to artificially place them on the above site if it is free; then, it follows the aeolian transport mechanism described in section 3.1.2, *i.e.* it will either fly away if the velocity is high enough, or fall back to its initial position (and thus changing nothing to the deposit).

From a practical point of view, there are two possible implementations:

- each frozen particle below a free site is ejected with probability ζ_p,

- or, if n_{frz}, the number of frozen particles on a site below a free site, is large enough, $\mathcal{B}(n, \zeta_p)$ gives in one evaluation the number of particles to be placed on the above site.

However, it is important to consider carefully the number of particle "candidate" to erosion. If we consider the following situations:



Counting only the upper layer of frozen particles would make us pulling particles among 100 of them in *(a)* (as $\theta_{frz} = 100$ and there is nothing above the solidified layer) and only among 1 in *(b)*. This produces a severe discontinuity in the erosion process as a totally different stock of "erodable material" is available when situations are more or less the same in both cases.

Therefore, it appears more correct to consider as erodable all the first solidified layer (*i.e.* θ_{frz} particles) plus the frozen particles in the underneath cell.

Hence, with this later option, we may consider differently situations *(a)* and *(c)*, even if they are very close (indeed, freezing one more particle in the rightmost situation would freeze the above site and therefore return to the same situation as the leftmost one). The solution is therefore to consider at most θ_{frz} particles as erodable ones (in fact we may consider less only if the particles are frozen directly above a solid cell which is not built from frozen particle, but from the initial ground configuration). This later choice is therefore equivalent to pull particles from a constant volume equivalent to a frozen site (θ_{frz} particles).

Eroding particle with eddies

It is reasonnable to erode more particles where the wind eddies are the more developped. Even if the eddies are smaller than the domain resolution scale, we have extrapolated theire influence with the fluid BGK subgrid model in section 2.2.3. As the local non-equilibrium momentum flux tensor has been computed in the Smagorinski process, we may re-use it here: instead of being (straigthly) a constant ζ_p, the local erosion probability can be multiplied by the maximum value of the non-equilibrium momentum flux tensor in the neighborhood. This idea has been implemented only on the 3D version of our code and gave better results.

3.1.5 Summary

In this section, we have introduced the different mechanisms, ruling locally the solid particles evolution. These mechanisms can be tuned with several parameters, mainly:

θ_{frz}	the number of particles to solidify a cell (frozen threshold
ζ_p	probality of erosion
$\mathbf{u}_{fall}$	falling velocity

These parameters, together with fluid paramters and domain sizes are summarized for all the $3D$ eperiment in table 3.1.

Algorithm 3.1.3 Particle deposition: freezing and solidifying steps

```
do $i = 1, q$                  // $q$ is the number of lattice directions
                               // for the particle model
  $p_{frz}(\mathbf{r}, t+1) \leftarrow p_{frz}(\mathbf{r}, t)$
  if $s(\mathbf{r} + \mathbf{c}_i, t)$ then
     $p_{frz}(\mathbf{r}, t+1) \leftarrow p_{frz}(\mathbf{r}, t+1) + p^{(i)}(\mathbf{r}, t)$
     $p^{(i)}(\mathbf{r}, t) \leftarrow 0$
  endif
enddo
if ($p_{frz}(\mathbf{r}, t+1) \geq \theta_{frz}$) $s(\mathbf{r}, t+1) = true$
```

3.2 Deposit results

The original part of this thesis is to add solid particles on a fluid using lattice gas techniques, thus its main validation must come from a confrontation to field results. However, the model scales (time and space) are not connected *a priori* from the fictitious world to physical scales, thus only the results can eventually give them.

Meanwhile, this lack of *a priori* scale connection offers the opportunity to explore (and expect valuable results) within a wide range of situations, by tuning different model parameters, such as the fluid viscosity, the probability of erosion ζ_p, the threshold number of frozen particle to solidify a site θ_{frz}, the initial landscape etc.

We will present, throughout this section, deposits in a very wide range of scales and situations, and we will confront them with real results:

- ripple patterns 3.2.1,
- trench filling 3.2.2,
- particle jumps length 3.2.3,
- different fences 3.2.4,
- two tent disposals 3.2.5,
- various road configurations 3.2.6,
- several mountain crest problems 3.2.7.

3.2.1 Ripples

From sand desert dunes to bottom of the sea ripples or snow field sastrugis, ripples grow almost everywhere (see figures 3.8 to 3.10). Within a very wide range of conditions, solid particles are not uniformly blown over a flat area, they tend to spontaneously self-organize in more or less regular macro-structures.

This problem is very elegantly stated: flat terrain, uniform initial conditions and regular particle input. Nonetheless, catching this macro self-organized phenomenon of ripples offers a great challenge. Classical approaches to model it are often dedicated to this question, and the final result is generally included in the model *a priori* definition [Werner *et* Gillespie 1993, Nishimori *et* Ouchi 1993a].

With the governing laws described in the previous section, ripples are not pre-stated. Therefore their spontaneous emergence would exhibit an evidence of validity in our model. Fortunately, in our model, ripples grow within a certain range of model parameters, as shown in the snapshot of figure 3.11. In this section, we will explore such results, try to show that they are not only a spurious model artifact, and see how the phenomenon is sensible to parameters variations (height, wavelength or ripple motion, depending on the fluid velocity, θ_{frz}etc.)

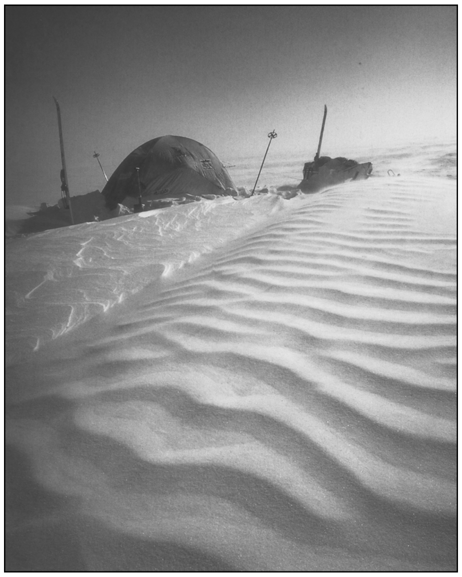
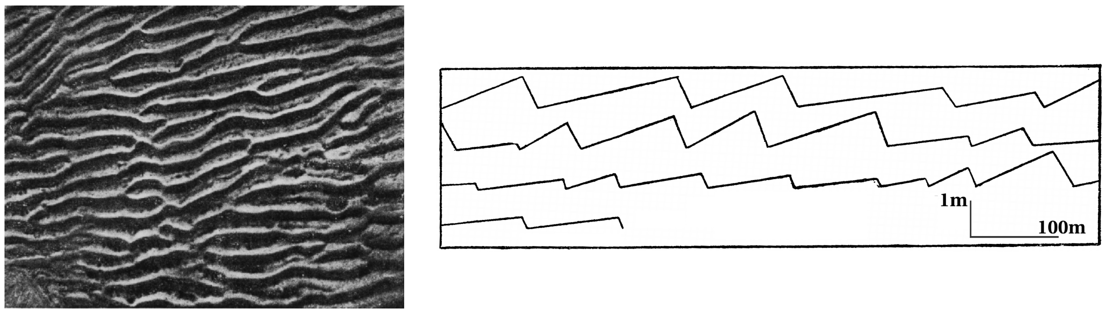


Figure 3.8: two kinds of snow ripple patterns: on the left hand picture, sastrugis eroded from coherent snow, where saltating particles can be seen as white smoke on the windward side (observed in Greenland); on the right hand pictures, ripples of fresh mobile snow (observed in Antarctic by [Ousland 1997]).



a) b)

Figure 3.9: in a), ripple-marks at Granges, Lancs, in sand of a somewhat tenacious kind washed by water. In figure b) profile of twenty-four aeolian sand waves near Helwan (Egypt), where one can note the irregular pattern of outdoor aeolian waves. Both were observed by [Cornish 1914].

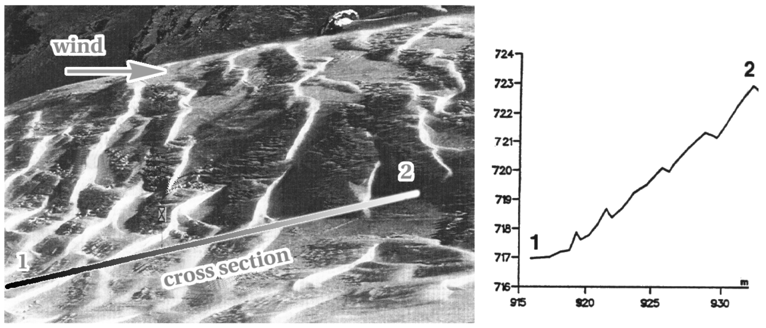


Figure 3.10: snow ripples in a mountain field, and, on the right hand figure, a cross section along the line $1-2$ line. Observed by [Castelle 1995].

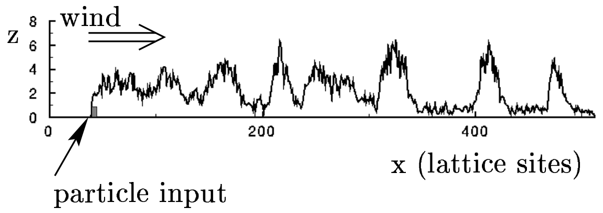


Figure 3.11: a snapshot of a typical deposit, after 50′000 time steps. At the beginning of the simulation, the domain is empty and the wind (fluid) is blowing from the left over a flat solid configuration. A solid particle input zone is set over several lattice sites at ground level (a rectangle of one cell high and one to ten cells in long), where the particle concentration is artificially set to θ_{frz}. With these border conditions, the only particles that may be a deposit must be eroded from this input area.

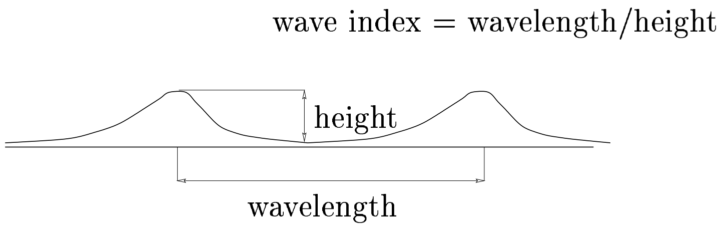


Figure 3.12: three basic ripple observables used in the phenomenon description. A fourth one, the motion velocity, can be extracted from 3.13.

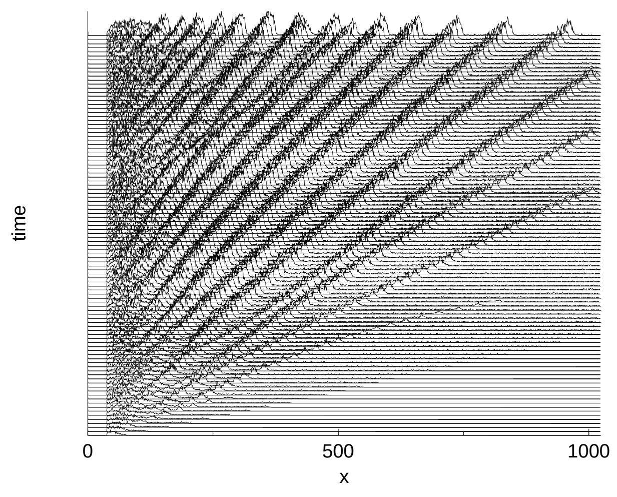


Figure 3.13: ripple deposits, with the same conditions as in figure 3.11, at different time steps. The lowest deposit (almost empty) is the first one. The regular translation of the patterns shows the ripple slow motion downwind. This figure also points out that small ripples travel faster than the larger ones, and that two ripples may merge to build a large one, as observed outdoor by [Cornish 1914]. The motion velocity of a ripple can be extracted from the slope of the line passing by its summit at various time steps.

Field similarity

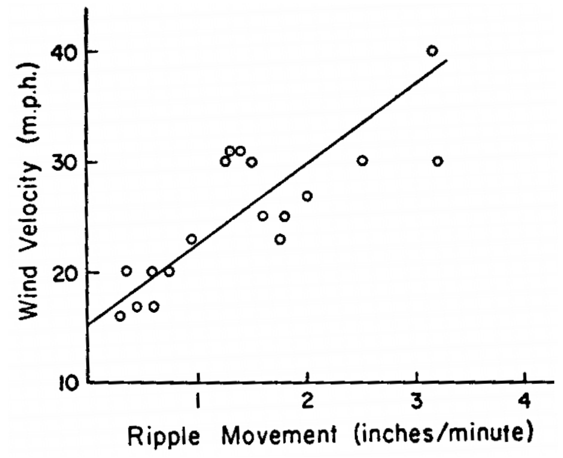
All outdoor ripples are obviously not similar to each other. Besides their shapes, their height or wavelength (see figure 3.12) may vary. Meanwhile, the wave index (ratio wavelength/height) provides a way to extract similarity between ripples of different scales.

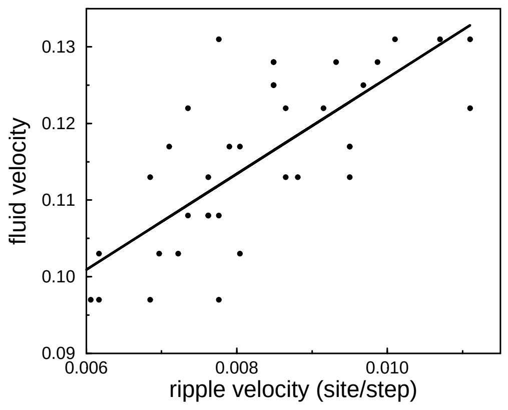
The shape of the ripple is not relevant in our 2D model presented here as we have not set any toppling mechanisms that would have recovered correct leeward and windward slope angles (see the similar approach by [Dupuis *et* Chopard 2000b]). Therefore the major similarity can come from the wave index and the behavior under the changing of some parameters.

Ripple motion

Sand or snow ripples (contrary to sastrugis) slowly move with the wind. The first step in our exploration was therefore to analyze this phenomenon.

The motion is also caught by our model, as shown in figure 3.13, where the slope of the line passing through a ripple top at various time steps gives its velocity (the steeper the slope, the slower the ripple is). We can therefore, in figure 3.14, compare the model ripple motion at different wind velocities compared to outdoor





a) b)

Figure 3.14: in a), plot showing sand ripples motion as a function of wind velocity measured 4 feet above surface observed by [Sharp 1963] at Kelso Dunes. In figure b) the same obsevation is plotted from numerical 2D simulations. The liniear fit is not good, but the correlation betwwen ripples motion and fluid velocity appears clearly.

measurements of sand ripples. It is interesting to note that, in both situations, the linear regression come to zero ripple movement at wind speeds of approximatively the half of the "interesting" range velocities (15.5 *mph* in the figure a), 0.06 in our numerical case b)).

From our simulations, we can also extract a relation between a ripple size and its motion velocity (see figure 3.15).

The wave index under-estimation

As stated in figure 3.12, the wave index of a ripple is the ratio of its wavelength versus its height. Extensive outdoor measurements have been achieved by [Cornish 1914] and can be summarized as:

- sand dunes averaged wave index is approximatively 18,
- snow (sand-like snow particles) one reaches 28,
- sand ripples in water average is 14; however, in some situations, it can be as low as 6.

For each simulation, this observable can easily be measured and offers a scaleless way to compare experiments. The numerical results, displayed in figure 3.16,

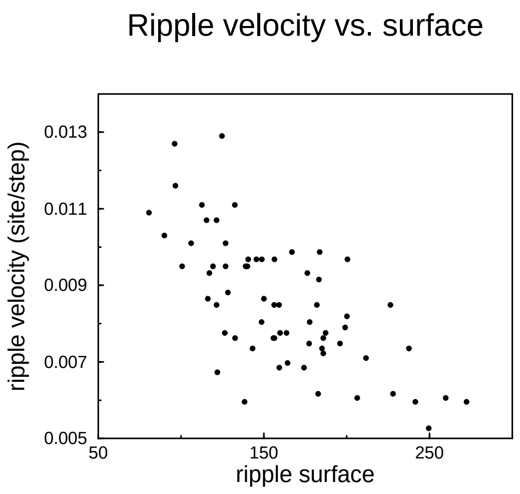


Figure 3.15: plot showing the ripple motion velocity versus its size (its surface, as the model is 2D). This graph confirms the *first glance* observation from plots such as figue 3.13 that the bigger the ripple is, the slower it moves.

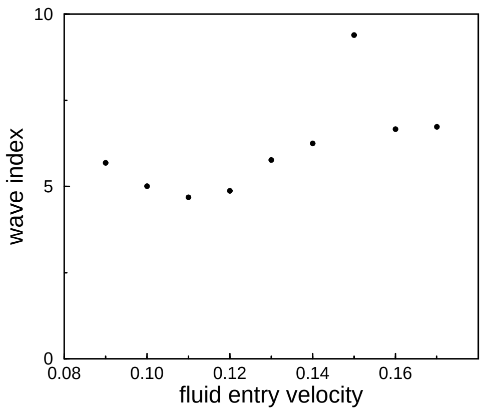
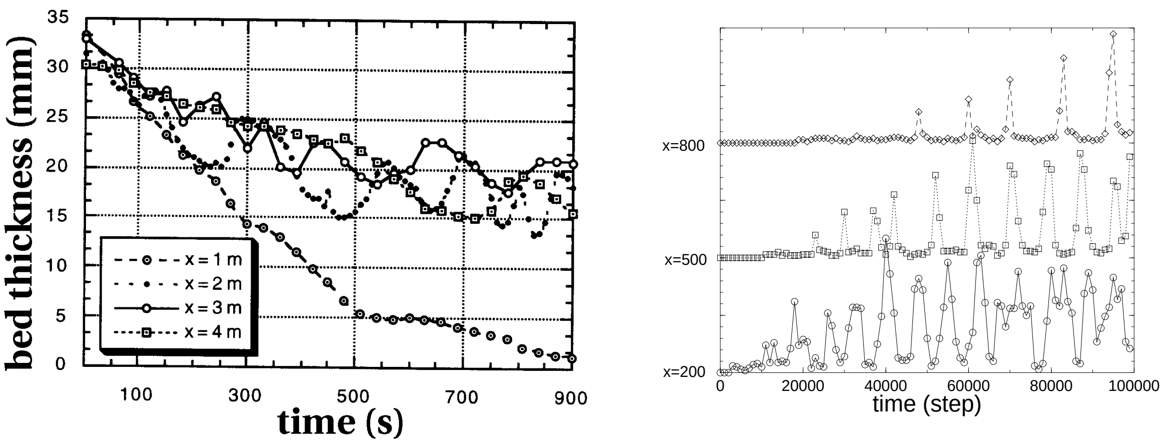


Figure 3.16: plot of the ripple wave index versus the fluid velocity; the model does not seem to correlate fluid velocity and wave index, and little litterature exist on this topic. In any case, the wave indexes are very low compared to outdoor ones (but at least larger than those simulated in a wind tunnel by [Martinez 1996]).



a) *b)*

Figure 3.17: plots of the bed height versus the time, at varous tunnel locations. Indoor wind tunnel results by [Martinez 1996] in *a)* and the same measurments from the numerical data displayed in figure 3.13. One can note that a certain regularity in the pattern is sooner recovered in the numerical experiment than in the wind tunnel one. This may be explained by the lower number of ripples developed in the latter one.

show a wave indexes shorter than aeolian outdoor recorded ones. Such low indexes can be found in tidal sand ripples, also observed by [Cornish 1914].

However, this phenomenon is even more emphasized in indoor wind tunnel experiments where the wave indexes are around 5, according to [Martinez 1996]. Therefore, the under-estimation of this factor seems to be recurent in both numerical and indoor wind tunnel simulations and may be related to their too coarse resolution of the reality.

To get larger wave-index, in the numerical model, one should lower the falling velocity $\mathbf{u}_{fall}$ to get longer flying trajectories. However, in our model, the intrinsic diffusion generated by the aeolian transport process (see section 3.1.2) blurs the precision of long range trajectories. Improving the precision of the trajectories can be done at the price of a better lattice resolution. However, as displayed in figure 3.5, a refinement of factor l increases the precision by a factor $\sqrt{l}$ but the requirment in memory by a factor l^d (and the computing times by a factor l^{d+1}).

Comparison with indoor wind tunnel experiments

As related in the previous paragraph, ripples have been reproduced in indoor wind tunnel by [Martinez 1996]. Although we compared wave indexes in both approaches, we may also focus on the more general development of the ripple throughout the simulation. A temporal evolution of the phenomenon is displayed in figure 3.17 and some observations can be achieved, such as the particle bed height at various locations, allowing comparisons bettween numerical model output and indoor results.

The simplest (?) ripple model

The purpose of this thesis is not to have an exhaustive look on the existing ripple models. However, it may be interesting to have a look at one of the simplest one published by [Vandewalle *et* Galam ress], with only particle pile height on each position along a $1D$ line. Three basic rules are applied at each time step:

1. a particle is randomly picked up and move to its rightwards neigbors ($\sim$ creeping);
2. if a pile height on a site is greater than that of its leftwards neighbour plus a threshold ξ, a particle is moved a given distance l away in the rightwards direction ($\sim$ saltation);
3. if a pile height on a site is greater than that of its rightwards neighbour plus a threshold ξ, a particle moves onto its rightwards neigbor pile ($\sim$ toppling).

Stating this rules on a $1D$ model quickly builds small ripples increasing in size [Vandewalle *et* Galam ress].

We have slightly modified this rule to get a $2D$ model (*i.e.* a 3D world). The two first rules are kept as-is, but the "toppling" mechanism must take into account the new dimension: instead of toppling only within the $x-$dimension, a particle can also fall from a pile onto one of its neighbors perpendicularly to the "wind" (thus in the $y-$dimension) if the neighbor's pile is lower by at least the thresold ξ; the direction of a topple is randomly drawn among the local possiblities. Ripples of growing size soon develop, as displayed in figure 3.18.

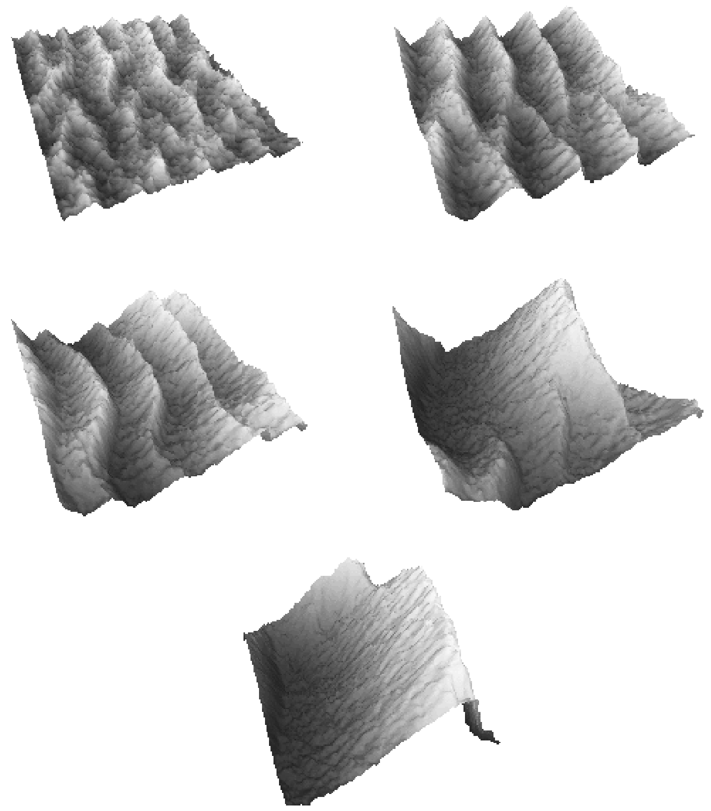


Figure 3.18: time evolution stages of the "simplest" ripple algorithm from [Vandewalle *et* Galam ress] adapted to a $2D$ deposit. The flat deposit grow in small ripples; the largest ones are caught by the smallest thus building an even larger ripples and so on.

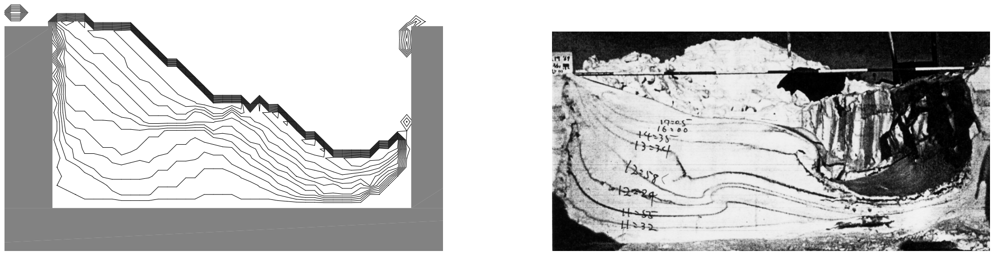


Figure 3.19: the problem addressed here is how a trench get filled in with time. Regular numerical deposit profile on the left can be compared with good qualitative accuracy to those found in the field by [Kobayashi 1972]; however, during the field experiment, the wind conditions were not stable, contrary to the numerical simulation.

3.2.2 Trench

A trench filling is observed at different times, and numerical results agree very well with outdoor measurments by [Kobayashi 1972] (see figure 3.19). It is interesting to note that the deposit does not grow in a trivial manner: several transport modes are involved: creeping for the left hand side deposit, saltation for the middle "dune" growing during the first stages, and some suspension for the small right hand side deposit. The two first modes are clearly identified in the next experiment.

3.2.3 Particle jump length distribution

The trench experiment proposed above shows how different transport modes are involved in particle transport and it is natural to make a step further by trying to quantify the particle jumps length distribution This experiment was acheived in outdoor conditions by [Kobayashi 1972] and reproduced numerically by our model with the layout shown in figure 3.20.

Both results agree very well, as shown in figure 3.21, for different wind conditions. Once again, it is interesting to note that, even if no separation into different transport modes was explicitely specified in our model, it intrisincally recovers reality with a good accuracy.

Experiments	Figures	$n_x \times n_y \times n_z$	$\mathbf{u}_{entry}$	τ	C_{smago}	θ_{frz}	ζ_p	$\mathbf{u}_{fall}$
Trench	3.19	$100 \times 3 \times 35$	.1	.5	.15	100	2.	-0.005
Jump length	3.20,3.21	$100 \times 3 \times 35$	.1	.5	.15	100	2.	$-0.005/-0.01$[a]
fences	3.22	$250 \times 3 \times 30$	.1	.5	.15	100	4	-0.01
Large fence	3.23	$93 \times 25 \times 15$	.1	.5	.15	100	3	-0.01
Tents	3.24	$140 \times 36 \times 20$	.1	.5	.15	100	2	-0.01
Roads (wind downhill)	3.25	$220 \times 3 \times 50$	.1	.5	.15	2000	2	-0.01
Roads (wind uphill)	3.26	$220 \times 3 \times 50$	.13[b]	.5	.15	2000	2	-0.01
Schwarzhorngrat's	3.28	$180 \times 3 \times 80$	.1	.5	$(0.15, 0.11, 0.2)$[c]	10000	1	-0.005
Marlennaz 1	3.30,3.31,3.32,3.33	$180 \times 3 \times 80$	.1	.5	.2	10000	1	-0.005
Marlennaz 3	3.34	$600 \times 3 \times 250$	.1	5	.25	10000	1	-0.005

Table 3.1: Parameter summary, for all the $3D$ numerical experiments displayed in this work. $n_x \times n_y \times n_z$ is the size of the domain (x is parallel to the main flow, y transversal and z vertical); $\mathbf{u}_{entry}$ the fluid entry velocity; τ the relaxation time; C_{smago} the Smagorinski constant for the subgrid model; θ_{frz} the number of particles to solidify a cell (frozen threshold); ζ_p the probabilty of erosion (in fact, this number is multiplied by the maximum value of the non-equilibrium momentum flux tensor in the neighborhood and returns and effective probability < 1); $\mathbf{u}_{fall}$ the falling velocity.

[a]Two falling velocities are used for two particle jumps length distribution measurements, when, in the field, the wind speed is actually increasing)

[b]THe fluid entry velocity is higher than in the previous experiment, even if the problem addressed here is more or less the same. However, before extrapolating the incidence of additional works (such as the slanted screens), one must at first recover the deposit over the free landscape.

[c]The goal in these three crests simultaions is to model the highest possible Reynolds number. Therefore, the Smagorinski constant must be adapted to the solid shape, to be the closer possible to the numerical blow-up limit.

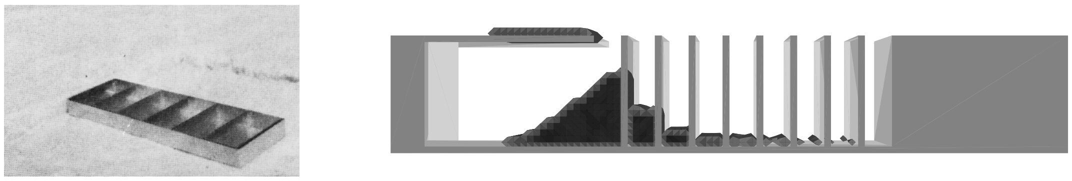


Figure 3.20: in order to quantify the distribution of particle jump lengths in the field, a device consisting in a suite of boxes aligned with the wind is set up as displayed in the left hand side of the figure [Kobayashi 1972]. On the right hand side is shown the same layout for a numerical experiment with the stock of particles captured by each boxes at the end of the simulation. The first box is digged under the ground to contain a larger amount of particles (in numerical simulations, it is useful to get the largest amount of transported particles to produce better statistics, when, in the field, a balance produces accurate enough measurements).

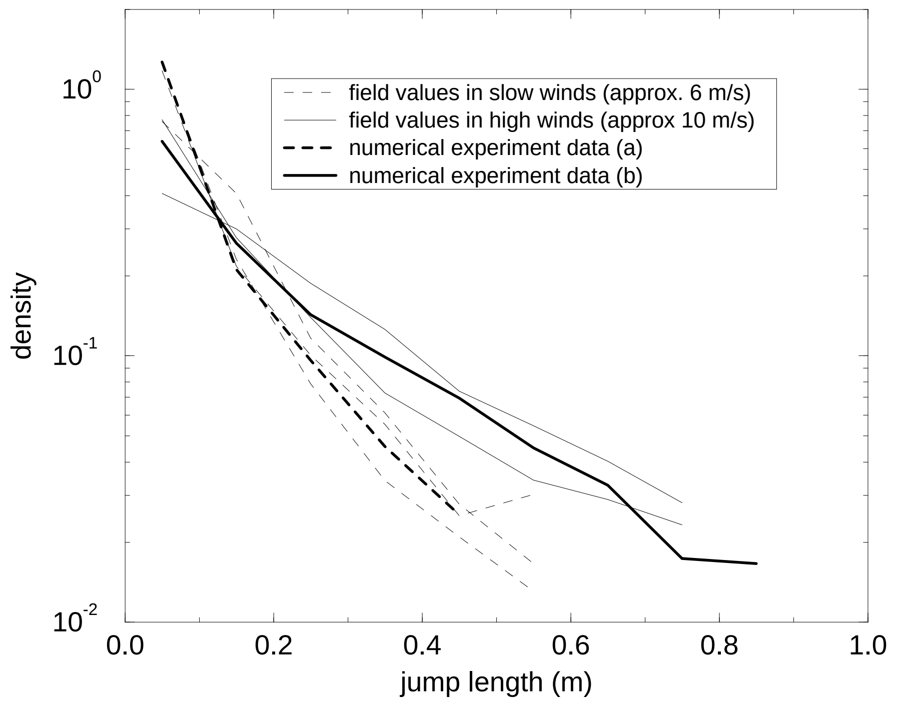


Figure 3.21: Density functions of particle distribution recovered in the different boxes presented in figure 3.20. Dashed line plots show results with a wind flux approximatively twice as low as for the solid lines results (in fact, the falling speed was multiplied by two in the numerical experiment). Thicker lines plot results from the numerical experiments, when thiner ones are extracted from field data [Kobayashi 1972].

3.2.4 Fences

The study of deposits around fences have been of an important practical interest as different kind of fences (full, porous - with horizontal or vertical planks -, with or without bottom gap) are used along roads or railways at many places.

We present in figure 3.22 two deposits, with or without a ground clearance to confirm the well known principle that a ground clearance lengthens the deposits [Castelle *et al.* 1992, Tabler 1980a].

In the field, the influence of a barrier is obviously of a finite distance. In the same manner, the deposits presented here are "steady" ones. It means that there are neither snapshots of time dependent deposits (that will looks different -and less presentable- after a some more computational time steps, like if it was to be growing in length forever) nor their length is imposed (thus shortened) by the domain size (it is the case, if doubling the tunnel length, with the same fence size, one obtains a longer deposit, proving the influence of the bondary). However, the deposit shape might be slightly oscillating around an average shape (in the order of 10% of the deposit length)

However, the length of a stable deposit is very dependent on the parameter choice, mainly the erosion probability ζ_p, compared to all the other simulations presented here. This can be explained by the fact that to recover a finite size deposit of size 20 times the fence height, we must model a very gentle leewards slope, based on a discrete regular lattice: this leads to a delicate balance.

Moreover, the windward small deposit often seen in the literature can easily be caught in many simulations. However, once we focus on getting a stable deposit, the parameter choice makes it vanishing.

A larger fence deposit is proposed in figure 3.23. The length of the deposit is only 6 to 7 times the height of the fence, but this is due to the relatively little height of the fence in regards to the tunnel cross-section (3 cells versus 15.)

3.2.5 Tents

As any obstacle raising in a snow drifting environment, tents soon get buried. If two tents are built pependicularly to the wind direction, a leeward deposit grows. A field observation and simultations (for two configurations) are shown in figure 3.24.

A better set up is proposed and should improve the situation, lowering the load of snow on the structures (however, confrontation with the field lacks and new experiments should be undertaken ...).

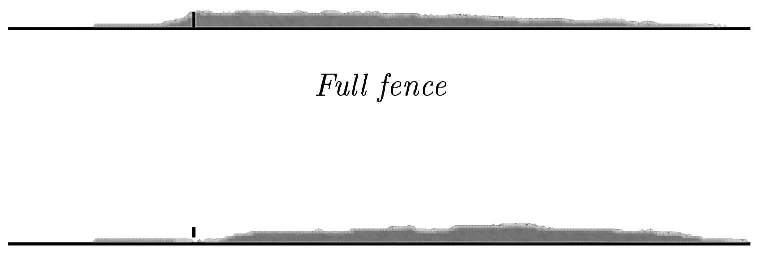


Figure 3.22: stable deposits (*i.e.* time stationary, not too much influenced by boundary conditions) around two type of fences. When a clearance is made under the fence and other parameters unchanged, the deposit shape is lengthened: it ranges here between 25 times (full fence) and 30 (bottom gap) the fence height, in good agreement with [Castelle *et al.* 1992]. This can be explained by two intuitive reasons: the gap prevent the fence from being buried and therefore loose its efficiency; the wind pattern is modified, as stated in figure 2.13 and is slowed down with a lower eddy activity.

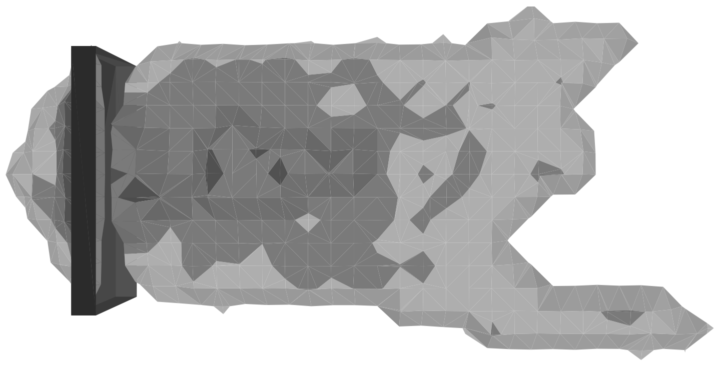
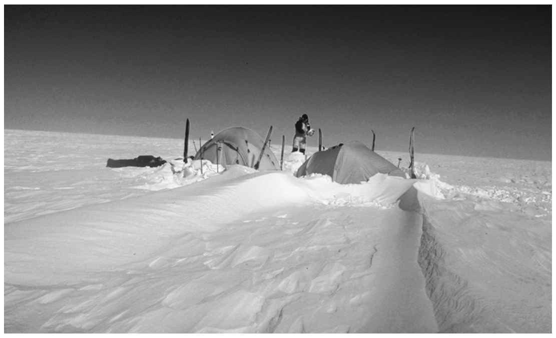
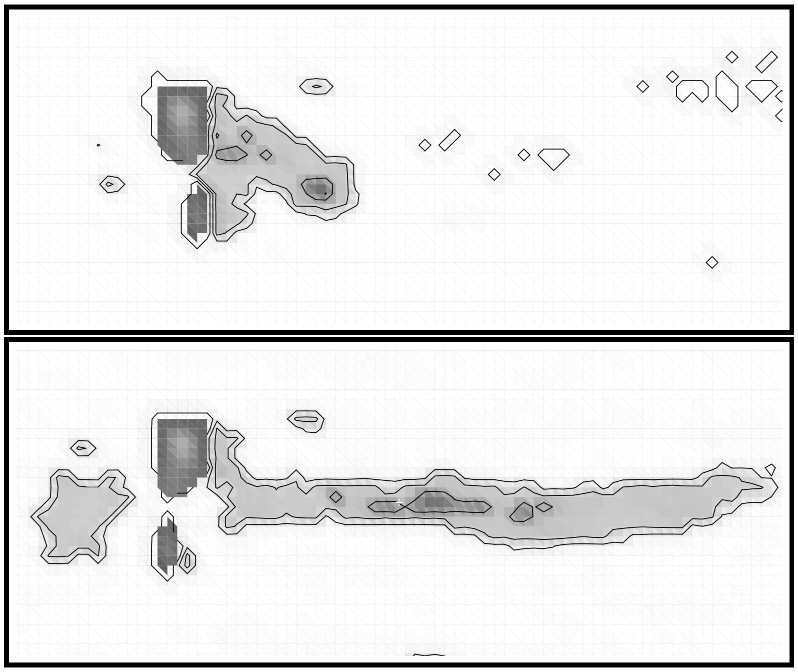


Figure 3.23: a real 3D deposit around a full fence, viewed from above; the darker the gray, the higher the deposit. The length of the snowdrift is too small, due to the small heigth of the fence, but the shape is in good agreement with [Castelle *et al.* 1992]. Moreover, some side deposits appear at some times at the leewards end of the snowdrift, in the same manner as in outdoor simulation around tents (cf figure 3.24). Domain size is $93 \times 20 \times 15$, and fence is only 3 site high but is enough to recover the deposit!



Deposit around two tents in the Arctic. Same patterns happen every days when the the tents are set up perpendiculary to the prevailing wind, as snow falls are rare, but drifting is continuous.



Two layouts of tents (same height, but one is thiner than the other) produces very different deposits patterns (view from above, wind blows from left, and the darker the figure, the higher the deposit - except for the two tents locations). The upper situation can be compared to the previous observation where the main deposit grows in between the tents, and both tents are buried on the leeward side; in the lower figure, another layout (both tents are aligned perpendiculary to the wind flux) produces a very different deposit shape, where the leeward snowdrift is thiner and longer, the smaller tent does not get buried and a small windward snowdrift grows.

Figure 3.24: snow deposit around two tents, set perpendiculary to the wind, in a flat area. Beside observations, simulations have been ran to determine the best possible set up.

3.2.6 Roads

Roads, as communication links, are of a crucial importance. Either alpine or nordic countries encounter tough problems in keeping these links open during the whole winter. More than snow fall (localized in time and regular in space), snowdrifts can close a road within a few hours, when several spots are buried because of their particular ground configuration in regards to the prevailing winds.

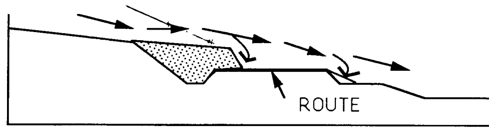
On flat spots, fences are erected parallel to the road at some distance (approximatively 20 times the fence height [Castelle *et al.* 1992]). However, many situations exist where the landscape appears to be more complicated, but the road has anyway to be protected.

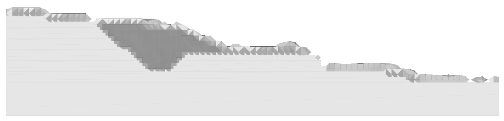
For two of these situations, extracted from [Castelle *et al.* 1992], comparisons where made between numerical simulations and field observations. Moreover, some adaptations proposed in the reference have been modelled, as well as new propositions imagined:

1. road follows a hill side, wind blows downwards and builds a snow drift (figure 3.25); the slope windwards the road can be flattened , or different slanted screens more or less cleverly disposed ;

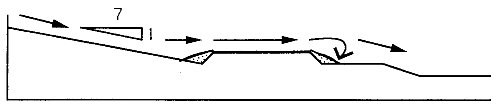
2. same situation, but the wind blows upwards the slope; initial deposit (figure 3.26) can be avoided flattening the leeward slope, but situation can be worsen building a slanted screen, with the emergence of a higher drift.

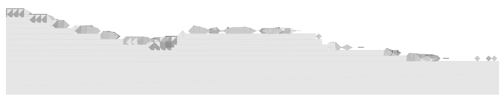
In the first case, a very good agreement is observed with literature results. The numerical model even allows to extrapolate the deposit building different kinfds of works. In the second case, the recovery of the small drift on the right side of the road is less clear in the numerical simulation; even in the reference, this drift is very thin, and may not be always observed in reality. Anyway, the deposit in the ditch is lightened when the leewards slope is flattened, in the same manner as in reality.



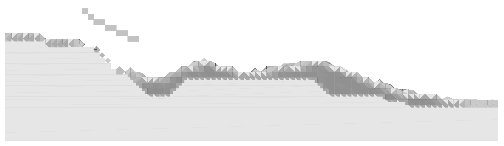


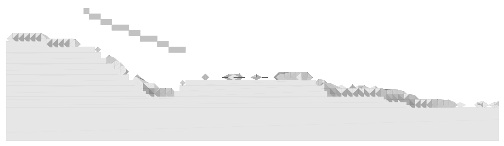
In both outdoor observation (left) and simulation (right), a snow drift blocking the road is present. The base landscape is of uniform gray, the darker a deposit cell is, the more particles are frozen on it.





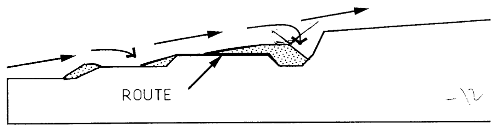
Flatening the windward slope produces, either in the field or in the simulations, a totally different deposit, as the drift is cancelled. However, this method can be hard to realize, either for practical or financial considerations.

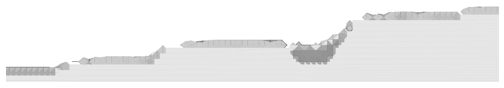




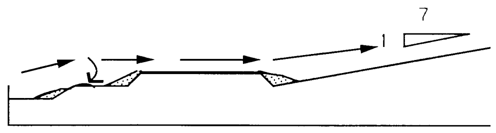
Two slanted screens scenarios are proposed. In the first case, the ditch drift is translated to the middle of the road; in the second case, a longer screen seems to avoid the problem.

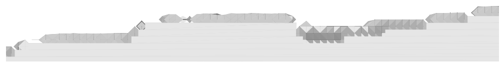
Figure 3.25: the road follows a hill side and wind blows downwards (from the left hand side), and avoiding the deposit emerging in the upper figures is the challenge [Castelle *et al.* 1992]. Simulations agree very well with outdoor observations and are used to propose more or less successful works.



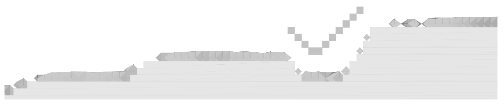


The snwow drift, observed outdoor is partially recovered by the numerical simulation.





The situation can be solved flattening the leewards slope. Observations and numerical output agree.



A way to avoid the drift coud be, at the first glance, setting up such a device as presented in this figure. However, according to the numerical simulations, the effect is unconvincing, as a non negligible drift grows by the middle of the road (the amount of frozen particles at this location is higher than in central simulation, as the gray is darker).

Figure 3.26: the road follows a hill side and wind blows upwards (from left hand side). The purpose of the experiments is to avoid the drift on the right part of the road.

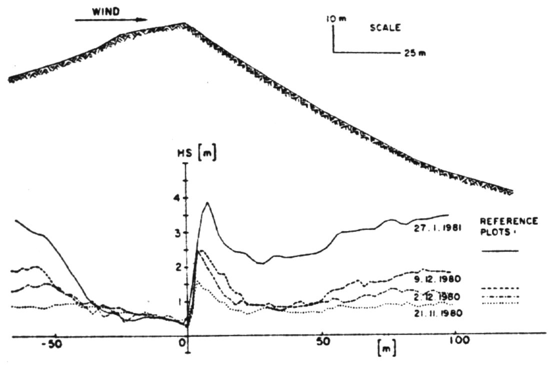


Figure 3.27: deposit evolution during the first two months of winter 80-81 on the Schwarzhorngrat. The monotonuous growth of this deposit (the same basic pattern is respected, even if the total amount of deposited material increases) makes this situation an interesting case study (if conditions are so irregular that the deposit grows in a totally disorganized way, predictions could hardly be made).

3.2.7 Mountains crests

At a higher scale, we may consider deposit problems around mountain crests. We have studied a litterature case, the Schwarzhorngrat from [Föhn *et* Meister 1983] in eastern Switzerland, and a more practical case in La Marlennaz (Valais, swiss Alps), where defence works are currently built.

Schwarzhorngrat

Although other crest cases exist in litterature, analysing a situation after long period often appear to be very irregular (many different conditions have ruled the system, different winds, snow falls with no wind, freezing of the upper deposit layer ...). From these points of view, the 1981 Schwarhorngrat study (recorded by [Föhn *et* Meister 1983]) offers three advantages:

1. the prevailing wind blows perpendiculary to the crest;
2. the deposit growth have been monotonuous during the recoding period, as displayed in figure 3.27;
3. measurments along three different profiles of the same crest (with steep, medium and gentle slopes) have been recorded, thus showing three deposits under the same input conditions.

For these reasons, we have choosen this case study. Our model parameters were calibrated on the middle slope profile, and ran with the same condition on the two other ones. Compared to the fence deposit parameters, the threshold number of frozen particle to solidfy a site, θ_{frz}, has increased from 10^2 up to 10^4, thus increasing the "actual size" of a lattice cell. From the fluid point of view,

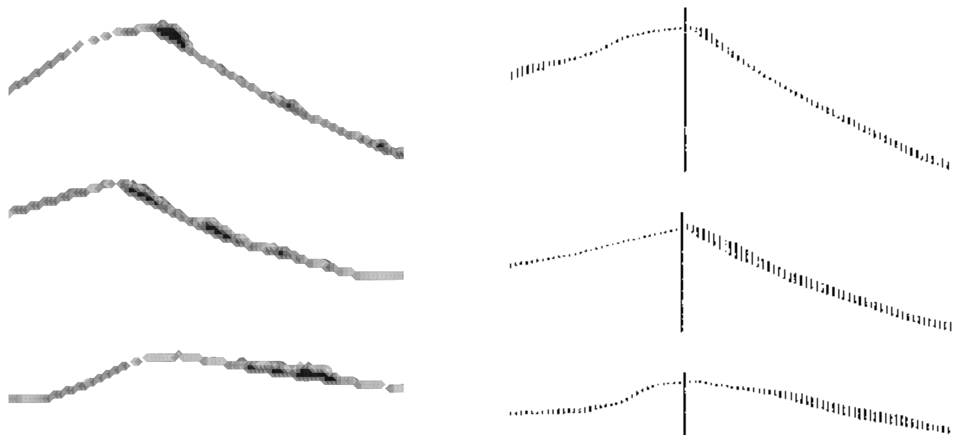


Figure 3.28: numerical results over three profiles at different positions along the Schwarzgrat crest, on the left, compared to field recorded measurments by [Föhn *et* Meister 1983] after the two first months of winter 80-81 on the right. Our model was calibrated on the middle slope profile, and ran with the same parameters on the two other ones. Space scale is given by the computer resources availability (*i.e.* the smallest possible cell size, but limited by both memory and time), but time scale is more difficult to approximate *a priori*: therefore, the simulation shall be stopped when the total amount of deposited snow reaches the experimental reference one. Anyway, as displayed in figure 3.27, the deposit in this case grows monotonuously, developing a basic pattern; it is this pattern to be recovered by the numerical model. To this respect, the results shown are in good agreement with reality.

C_{smago} has been decreased to its lowest value possible without a numerical blow up (C_{smago} was different for the three profiles, as it can be tuned lower when the profile is smoother).

The results of the numerical model over the three crests are confronted to outdoor observations in figure 3.28 with a very good agreement.

La Marlennaz

A more challenging situation occurs in the Marlennaz case, as no deposit records exist. However, in this site situated in the swiss Alps near the Verbier ski resort, different defense works are tested in real conditions by the Canton du Valais (service des forêts et du paysage). When nothing is done, a cornice grows at the crest level and later trigger avalanches that may cause serious damages to habitations, as it has already happened in the past.

To retain the mantle of snow, avalanche barriers have traditionnaly been built on the whole avalanche release slope, for more than one century. However, another approach (at least a complementary one) can be undertaken by trying to fight

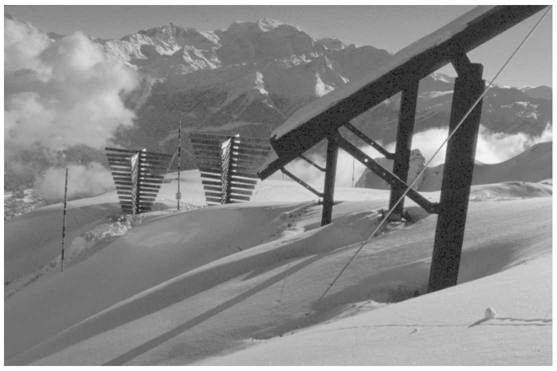


Figure 3.29: two different types of defense works, aimed at modifying the wind pattern and therefore the snow deposit at the crest location. Foreground, a slanted screen accelerates the wind parallel to the leeward slope and prevent the deposit. In the middle distance, the wind-veering will higher the wind turbulence leewards and disorganize its action in the area (the widen shape is important as it also accelerates the wind at the ground level).

against the cornice growing. Instead of regular mining with explosive during the whole winter, the idea is to build works in order to modify cleverly the wind flux, such as:

- fences, to store snow windwards the crest,
- slanted screens[1], to acclerate the wind parallel to the slope, thus breaking the cornice,
- wind-veerings, vertical crossbars, to alter the wind flux pattern.

The question is therefore to place these works in the most efficient position to avoid the emergence of the cornice, and to see how side effects (growing of an important wind slab, for example) can emerge.

Therefore, it was a natural application (at least a challenge) for our numerical model. Simulations were performed over two landcape profile, called profiles 1 and 3.

Marlennaz - profile 1

In the *profile 1* situation, where a slanted screen has already been built, simulations were ran with and without the work: snow deposits (figure 3.30), wind vorticity (the wind velocity field rotational, in figure 3.31) and flying particle

[1]The french term is “toit buse”, something like “shaft roof”, and the term “slanted screen” is a personal traduction. In the same manner, “wind-veering” is an approximative translation for “vire-vent”.

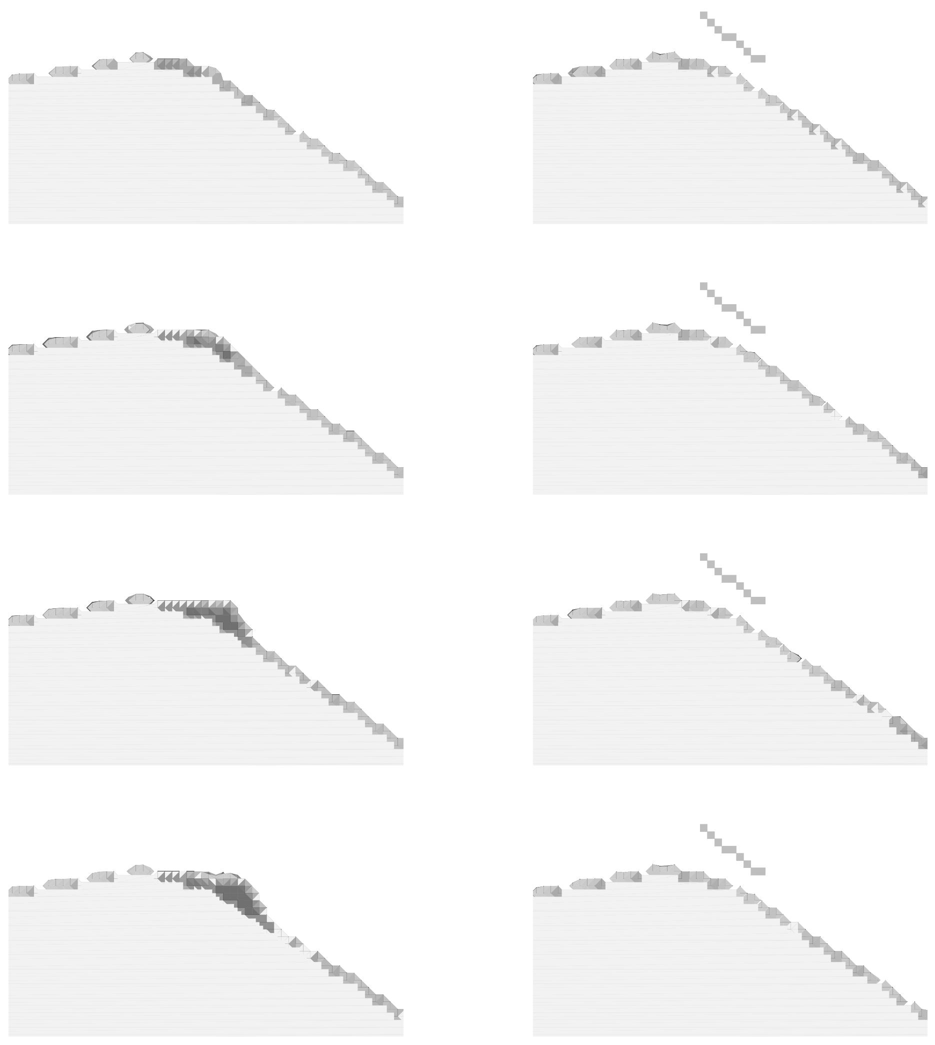


Figure 3.30: *La Marlennaz - profile 1.* Computed deposits the crest are plotted every 6000 time steps, with (right) or without (left) a slanted screen. As observed outdoor, the screen is efficient to limit the growing of the cornice. In such a simulation, the lattice resolution is coarse, and cell size effects are visible (such as the deposited particle concentration nearby each landscape stair step); however, averaging the deposit over the $x-$dimension lower this model artefact.

distribution (figure 3.32) have been compared. Later, different slanted screens angles and locations along the profile have also been tested (figure 3.33).

Marlennaz - profile 3

This profile is more challenging from two points of view:

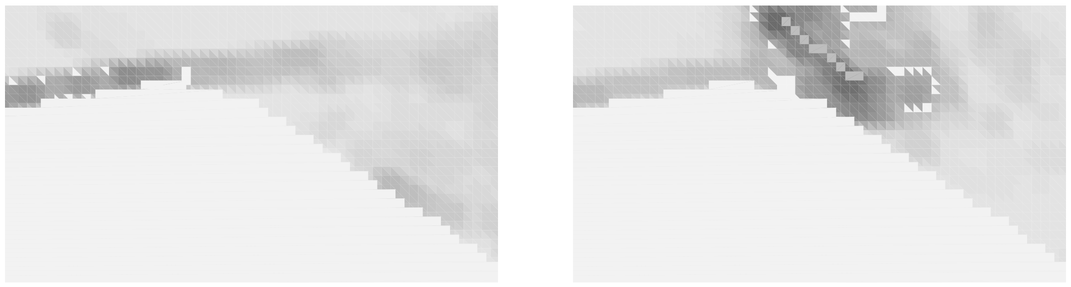


Figure 3.31: *La Marlennaz - profile 1.* Vorticity plot, where one can see the action of the slanted screen (the darker, the higher the vorticity).

1. the interesting part of the profile is the main slope break; however, it is located leewards a small little crest, whose influence is hard to estimate *a priori*;
2. no works have yet been built (except regular avalanche barriers in the slope), but some are planned in this area: therefore the question of what and where to build them is of a practical importance.

The computed deposit over the naked crest is compared with two slanted screens configurations in figure 3.34. According to the model results, a slanted screen at the slope breaking would clearly get rid of the cornice.

3.2.8 Simulations considerations

As for a laboratory one, such a numerical wind tunnel, as presented in this thesis, requires some skills from the user. Indeed, the model is a fictitious representation of the real world, thus some transposition rules might be applied, such as the Iversen criterion in indoor wind tunnel [Castelle 1995].

On the one hand, these skills shall be generic throughout the variety of problem faced, as a garanty to tackle a new situation (*i.e.* when no calibrations had previously be made) with the best chances of success . On the other hand, these skills are some recipes developped by the user, mainly to remain in the field of credibility (as most of the model parameters are not quantitatively linked to physical values, it is not obvious to determinate their interesting ranges of application). To "understand" such a numerical model behavior, it was very comfortable to rely on a 2D simulator with a fully interactive interface, such as presented in figure B.1 before exploring the 3D world.

3.3 Conclusions

In this chapter, we have presented the results, thus the validation of our numerical model. We have confronted it a very wide range of problems, which are tradi-

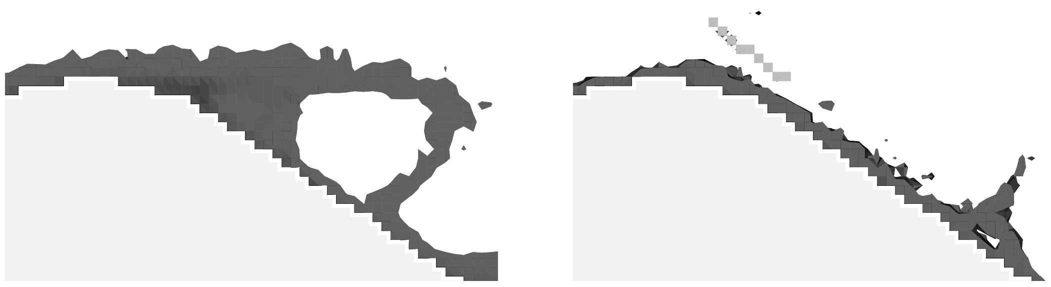
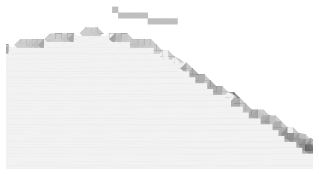
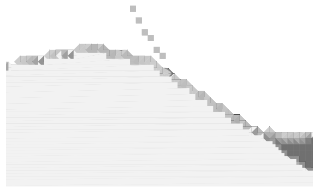


Figure 3.32: *La Marlennaz - profile 1.* Typical snapshot of the flying particle concentration around the crest. In the left hand experiment, the presence of a well established vortex at the slope break location induces the cornice, when the slanted screen (right figure) accelerates the wind and disorganize this vortex. However, in this latter case, the higher concentration of particles at the ground level increase the risk for a wind slab to occur, mainly by secondary slope breaks.

(a)



(b)



(c)

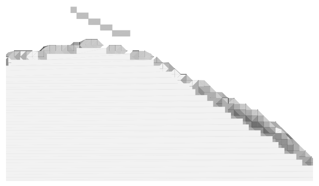


Figure 3.33: *La Marlennaz - profile 1.* Computed deposits with different slanted screen configurations. Obviously the first two are either not enough or too much inclined (*(a)* and *(b)*), but the aim of the simulation is to show how the model reacts under such conditions and how the most efficient angle could have been predicted (anyway, it was initially set into the most intuitive configuration). Experiment *(c)* is more interesting as, from a practical point of view, a slanted screen could have been set up in such a configuration (*i.e.* upper in the slope, at the level of the first slope break): it would have breaken the cornice, but have generated a inopportune slab down the slope as a side effect.

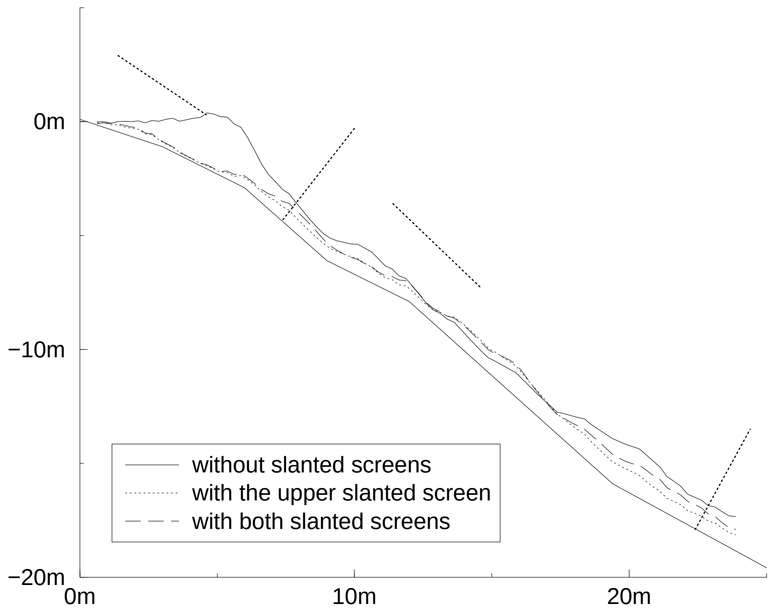


Figure 3.34: *La Marlennaz - profile 3.* Different simulated deposit over the crest shown in the inner figure. The main depostit develops leewards the slope breaking, where a large cornice appears when no works are set up. With a first slanted screen (thick solid line), it totally vanishes, but a slightly thicker slab is produced around $x = 15$ meters. From the simulations (as well as from intuition), it is not possible to get rid of the second deposits. However, it can be an interesting information for the placing of traditionnal avalanche barriers (the actual ones are indicated by the thick dashed lines, and as they are built from iron net, there were not considered as modifying the wind flow).

tionnaly treated by dedicated, totally different approaches. A key features is how the smallest, like the largest scale problems can be adressed. Ripple formation could be caught, but so were deposit prediction around alpine crests.

The capacity of our numerical tool to answers correctly to these questions shows how the lattice gaz technique (applying local simple rules on a fictitious discrete world) is very well suited to the complex system of the erosion/transport and deposition of snow by wind.

Moreover, as we will see in the following chapter, short computing times (most of the crest and the roads problems were computed in less than fifteen minutes on a parallel computer) allow an interactive use of the simulator. Therefore, it is comfortable to investigate new situations, such as how variations in the landscape (fences, slanted screens ...) influence the deposit.

Chapter 4

Implementation

Lattice Boltzmann models for the fluid or solid particle dynamics, such as presented in the current work, are very well suited to be efficiently implemented on a computer. Contrary to classical CFD codes, there are neither complex systems of differencial equations (where each new component complexifies the whole system), nor implicit iterative method involving global and expensive computations (inverting matrix etc.).

Although the cellular automata approach is rather simple (local computation and nearest neighbours communications on a regular array structure), we will focus in this chapter on the differents aspects of an efficient implementation:

- high level language choice,
- program structure,
- parallelism,
- hardware.

4.1 The programming language: Fortran 90/95

In scientific and high performance computing, two major languages are currently available. Both have proven to be built on reliable basis and are developped to fullfill up-to-date programmers requirements in the field. We are interested in: C++ and Fortran 90/95.

In a first subsection, we will present a short overview of both languages; we will later show why our choice is a good one.

4.1.1 C++: an overview

C has emerged with the Unix operating system (and Unix has emerged with C). Analysing the success of C would require at least a whole chapter, however, we can briefly state that its power, thus its success, has been due to this strong link with Unix roots, the possibility to access virtually any features of the machine, its ability to simply manipulate simple or complex data structures (sometimes in ways more readable for a compiler than for a normal human being).

Designed as a high level language in the seventies, C can be more and more considered as a low level language (a hacker language, where the best is possible, in terms of efficiency and power, as well as the worst, in terms of cryptic sources, absence of security etc.) in regards to modern languages criterias (object philosophy, security, software engineering). C has therefore evolved to C++.

The growing complexity of codes has forced the programmers to bypass the slicing of a program only into procedures and to imagine more complex structures: *objects*. Their major features can be summarized as:

- an object is an entity containing a data structure, but also a set of procedures (methods) to manage this structure (to create, destroy, access, manipulate it);

- inheritage and genericity are two powerful concepts allowing to base objects upon other ones (for example, an object *matrix* can be built, with several tools - addition, multiplication, but also inversion, transposition - containing object specified elsewhere - either *float*, *integer*, *complex* or any other objects for which some basic *methods* have been defined);

- objects can be developed separately, providing a comfortable abstraction level and facilitating the coarse grain software engineering of a large product, *i.e.* maintenance and modification;

- because of this independance, an object can be (at least theoritically) reusable, *i.e* developped elsewhere, downloaded, and used *"as is"*.

Of course, C++ is not the only *object-oriented* programming language. Java, for example, is closer to the pure object "philosophy", but C++ takes advantage of its C roots and is much more adapted to intensive high performance computing.

4.1.2 Fortran 90 and 95: an overview

Since Von Neumann's UNIVAC in 1951, some scientific researchers have seen computers as modelling tools. Once they got sick of punching cards with machine code, they designed a higher level programming language to fullfill their needs: massive computations, differential equations modelling, linear algebra. Fortran was born.

Fortran IV, 64 and 77 have come to life. Nowadays, Fortran 77 is often seen by the computer scientist community as the latest re-incarnation of a pagan divinity, and their opinion ranges from condescendance to pure nausea. Indeed, standard

Fortran 77 can hardly be regarded as a modern programming languages: no dynamic memory allocation, use of labels, no data structures, no function overload, fixed format source code (the famous 7^{th}-72^{nd} columns), no programming structure possible (except linear list of procedures).

However, this bad opinion can be explained (thus forgiven) as based on ignorance, and hard living myths. Since the begining of the nineties, the Fortran standard has considerably evolved: Fortran 90 was synthetized (later Fotran 95, with minor changes)[1]. Its major new features are:

- dynamic memory allocation;
- a program can be split into modules (see section 4.2.2 for details about our presented program); a module contains data, functions and subroutines; all the subroutines of the program using a given module will share its ressources;
- array manipulations are *intrinsic* (if A, B and C are multi-dimensional arrays, one can straightforwardly write $A = 3$, $A = B + C$, $\mathrm{sum}(A)$, as well as many intrinsic functions or subroutines);
- procedure overload (a procedure can be called with different kinds of arguments; depending on these argument, a dedicated procudure will be called at compilation time);
- structured data types, pointers;
- basic operators $(+, -, <, / = \ldots)$ can be redefined for user data types.

Fortran 95 has only tiny modifications, mainly the suppression of some obsolescent features remaining from Fortran 77.

4.1.3 And the winner is ...

Fortran 90/95. The tremendous suspense was anyway already killed by the section title.

We may now try to justify this choice, precising that the decision of programming in Fortran 90 is valid for the class of problem addressed in the current work: lattice gas models.

As stated at many different places throughout this thesis, the cellular automata method represents the simulation domain as a regular array and splits each time step in two stages:

[1]As Fortran 90 handbook, one may check the very good [Metcalf *et* Reid 1999] by one the Fortran forum gurus, or the more pedagogical (unfortunately written in french) [Delannoy 1997].

- **computation** of the cell evolution, based on local informations (fluid and particles densities within each lattice direction, solid configuration around the cell);
- **regular communications**, with the nearest neighbors.

A natural and straight forward implementation is therefore to store the information in a multi-dimensional array (for example $9 \times n \times n$ for a BGK model D2Q9 over a $n \times n$ square domain). The language intrinsic array feature are therefore comfortable. Moreover regular communications are also intrinsic in Fortran 90 (*cshift*).

Obviously, it is possible in C++ to define *matrix* object and methods to manipulate them in the Fortran style. However, Fortran 90 offers these features in standard. Moreover, any computations over scalar variables (trigonometric, bit manipulation, conversions) can be applied on arrays at no programming cost, and many array procedures are available (spread reduction operation - assignement, *sum*, *maxval* - globally or within a given dimension, array packing, communications).

Once again, all these features can be implemented in C++, but as they exist in standard in Fortran, the compiler may be able to call them in the most efficient way. Often, in comparisons between Fortran 90 and C++, it is shown how inadequate it is to try to model object features with Fortran, and pointed out how easily the intrinsic features of Fortran can be incorporated in C++. This is definitely true, as Fortran 90 has never claimed to be a universal language, such as C++, but only an efficient one in its field. And this aim has been reached

From another point of view, Fortran 90 array notation is intrinsically parallel: when one writes $A = B + C$ (A, B and C are arrays), the code can easily be parallelized by the compiler (arrays are therefore shared between several processors only with add-on compiler directives, for example with High Performance Fortran or on a shared memory machines with OpenMP).

Last but not least, the main usage of Fortran is for intensive computations, therefore the efficiency of compiler soon appears to be an important point on the market for manufacturers. As the language is relatively simple, they can achieve very good results. The simplicity of the language is also a key point, as it can be learned rapidly and writting source code is simpler than C++.

C++ is by many means far more powerful than Fortran 90/95, as it can address a much wider range of applications, and is much more suited for a high level software engineering design of a program. However, this generality (combined with some lack of security due to the C underlying layer) results in far more complex tool. Our lattice gas problem is perfectly and very elegantly covered by the Fortran 90 range of application, and this justify our choice [2].

[2]**NB:** a lattice gas problem can be addressed in a totally other manner with the object approach: a cell is an object with all the local information, fluid and particle evolution are

4.2 The program structure

Once Fortran 90 has been elected, we should present how the data is stored to represent fluid, solid particles and solid over the domain.

The language offer the possibility of structuring the information into modules containing:

- data,
- procedures.

Therefore, the global application can be represented as an oriented non-cyclic graph of modules. Each subroutine, program or even module *using* a pre-defined module can access its data and execute its procedures.

4.2.1 Notations

We will focus on a 3D domain of size $n_x \times n_y \times n_z$, where n_x is supposed to be the largest dimension (this information will come to be important in the section 4.3); we will try to present the method in the most generic way, so it can easily be reduced to a 2D domain. Most of the variables presented here have already been presented in the previous chapters, however we may repeat them here in a more practical fashion, to offer a better understanding of the implementation. Some detail enumeration may sometimes look like a tedious recipe book, however these recipes have been extracted after many rewriting of Lattice Boltzman codes.
NB: arrays are labeled in the Fortran fashion, *i.e.* $a(n)$ contains values ranging from $a(1)$ to $a(n)$, and $a(m\!:\!n)$ values from $a(m)$ to $a(n)$.

4.2.2 The modules

Domain module (*world*)

The variables depend on the lattice topology:

- d, *integer*: lattice dimension (3 in this case);

methods, neighbors are pointers to other cells (not necessarly regularly arranged in the computer memory). In this approach, only cells where something is happening (not inside the solid, for example) need to be allocated (this saves memory space at the first glance, but the extra-information needed by each cell is not negligible). The CPU time needed for this representation of a simple lattice Boltzmann model is twice as long as the one spent with a more classical representation [Dupuis *et* Chopard 1999]. However, complex features such as load balancing maybe implemented in a more easy way, as good object code is definitely easier to maintain once the project has reached a certain size.

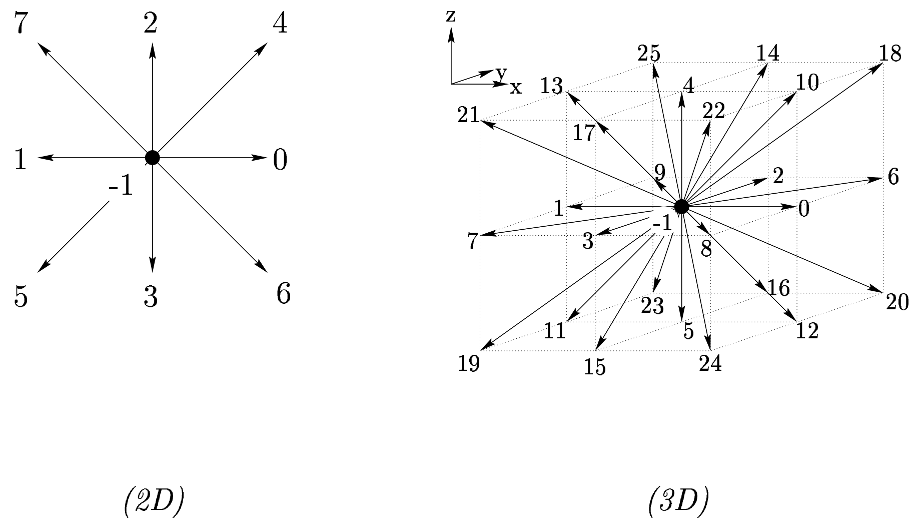


Figure 4.1: labelling methods for 2D and 3D lattices. It is more efficient, for manipulating data, to follow some stresses: directions $2i$ and $2i+1$ are opposite to each other; vectors of norm 1 are labelled at first, then those of norm $\sqrt{2}$ and at last $\sqrt{3}$. In the $D3Q19$ gas model, only directions from -1 up to 17 are used, but solid particle densities are attached to all the 27 directions.

- q, *integer*: maximum number of neighbors for each cell in a cartesian system: $q = 3^d - 1$ (as within each dimension, a vector coordinate can be one of the 3 values -1, 0 or 1, and $(0,0,0)$ is not a lattice vector);

- q_f, *integer*: number of directions for the fluid lattice (18, if we use the $D3Q19$ BGK model);

- $\mathbf{c}$, $real(-1:q-1,d)$: where $(\mathbf{c}(i,1), \mathbf{c}(i,2), \mathbf{c}(i,3))$ ($\mathbf{c}(i,:)$ with Fortran 90 notation) is the i^{th} vector of the lattice; for a further easier and more homogeneous use of these variables, it may be useful to stress the following tricks, displayed for $2D$ and $3D$ cases in figure 4.1:

  - vector -1 ($\mathbf{c}(-1,:)$) is the $\vec{0}$ vector (corresponding to the "resting" particles),
  - vectors $\neq \vec{0}$ are labelled from 0, so that index ($i \bmod q$) is a always meaningful direction,
  - in $3D$, directions $2i$ and $2i+1$ should be opposite to each other; both these latest techniques facilitate "bounce back" features implementation (this technique can also be used, although it is easier to label the direction clockwise or counterclockwise),

  - the first q_f indexes (from 0 to $q_f - 1$) point towards directions of the fluid lattice (for example in the $D3Q19$ fluid model, each node has 18 neighbors while the underlying 3D lattice offers 26 of them); therefore all the local momentum and mass sums can be performed only on those indexes (see equations 2.3 and s 2.4);

- $\mathcal{S}_{ind}$, $integer(d, -1{:}1, 3^{d-1})$: where, for example the indexes $\{\mathcal{S}_{ind}(2,-1,i)\}_{i=1\ldots 3^{d-1}}$ are the labels of the vectors pointing, in dimension 2 ($y-$dimension), towards direction -1; this list representation can be built automatically (based only on $\mathbf{c}$) at once, and will be useful for the streaming procedure; once more, it may be clever to give the priority in the list order to the fluid lattice indexes (in the $D3Q19$, only 5 fluid densities are to be streamed along each direction);

- $\mathcal{S}_{bitmask}$, $integer(d, -1{:}1)$: carries the same kind of information as $\mathcal{S}_{ind}$, but on a different manner: the i_q^{th} bit of $\mathcal{S}_{bitmask}(i_d, i_s) = 1 \Leftrightarrow \mathbf{c}(i_q, i) = i_s \Leftrightarrow \exists k$ with $\mathcal{S}_{ind}(i_d, i_s, k) = i_q$; This representation turns out to be useful when some information on the i_q^{th} bit will be related with the direction i_q^{th}. For example in $2D$, with the labeling given in figure 4.1, indexes of vectors point in the $x+$ direction (dimension 1) are 0, 4 and 6: $\mathcal{S}_{ind}(1,1) = (0,4,6)$ and $\mathcal{S}_{bitmask}(3,1) = 2^0 + 2^4 + 2^6 = 81$.

The module *world* contains several procedures:

- `world_init` allocate all the arrays according to the domain dimensions n_x, n_y and n_z, and compute $\mathcal{S}_{ind}$, $\mathcal{S}_{bitmask}$ etc. from informations contained in the $\mathbf{c}_i$'s;

- and all-purpose utility functions and subroutines that can be called by any of the following modules.

Fluid module (*fluid*)

Variables are split in computed fields over the lattice:

- f, $real(-1 : q_f - 1, n_z, n_y, n_x)$: fluid densities among each lattice link, where $f(i_q, i_z, i_y, i_x)$ contains the value of $f_{i_q}((i_x, i_y, i_z), t)$ from section 2.2.2 and time t is the current simulation time step; we will see in section 4.3 the reasons why we have chosen this index order, *i.e.* n_z first (as, for example, $(n_x, n_y, n_z, -1 : q_f - 1)$ can appear at the first glance more reasonable);

- $\mathbf{u}$, $real(d, n_z, n_y, n_x)$: fluid velocity at each lattice node, which shall be computed at each time step;

and user defined parameters:

- κ_{uentry}, *logical*(d): set the sides from where to accelerate the fluid ($\kappa_{uentry}(1) \Rightarrow$ fluid is accelerated from the $x = 1$ slice, $\kappa_{uentry}(2) \Rightarrow y = 1$ and $y = n_y$ slices, $\kappa_{uentry}(3) \Rightarrow z = n_z$ slice); usually, fluid is accelerated from the entry side ($\kappa_{uentry}(1) = .true.$), but in some situations, it may become useful to accelerate it also on other borders;

- $\mathbf{u}_{entry}$, *real*(d): velocity value to which the fluid must be forced to in the dedicated areas defined by the previous variables (for example, to enter an horizontal velocity with speed 0.1, we may set $\mathbf{u}_{entry} = (0.1, 0, 0)$);

- τ, C_{smago}, the relaxation time and subgrid parameters giving the fluid viscosity , described in section 2.2.2.

The main procedures are:

- `fluid_init`, to allocate the module arrays;

- `compute_u_`ρ, to get the local velocity vector and density for a cell, depending on the local f distribution;

- `compute_`f^{eq} for the local equilibrium distribution;

- `fluid_stream` to stream the fluid densities along the lattice links (this procedure is only several circular shift call base on the indexes defined in $\mathcal{S}_{ind}$ on a single process program version, but it will become more tricky on a multi-process parallel version);

- and some utility functions, to check the total amount of particles, momentum throughout the system.

Solid particles module (*part*)

In the same manner, we may also split the variables described in section 3.1 among runtime computed fields:

- p, *integer*$(-1 : q-1, n_z, n_y, n_x)$: defined in the same manner as f array, contains the integer number of particles travelling along each lattice link;

- p_{frz}, *integer*(n_z, n_y, n_x): the number of frozen particles per site (cf. section 3.1.3);

and a long set of user defined (and possibly time evolving) parameters:

- $\kappa_{p-cyclic}$, *logical*(d): defines if the particle streaming is cyclic among each dimensions (for example, if $\kappa_{p-cyclic}(2) = .true.$, the particles pointing in the $+y$ direction in the $y = n_y$ slice would re-enter the system at the same position in slice $y = 1$; this is useful to model a cyclic system in the $y-$dimension);

- $\mathbf{u}_{fall}$, $real(d)$: the particle falling velocity (cf. section 3.1.2); it is interesting to prefer a d-dimensional vector to a single vertical component, as an α-inclined tunnel can easily be modeled by letting the falling speed of module u_1 be $(u_1 \sin\alpha, 0, -u_1 \cos\alpha)$;

- θ_{frz}, $integer$: the number of frozen particle before solidifying the site (cf. section 3.1.3);

- Λ_p: list of hexahedron $(i_1, i_2) \times (j_1, j_2) \times (k_1, k_2)$ $(\{(i, j, k)$ — $i_1 \leq i \leq i_2$, $j_1 \leq j \leq j_2$ and $k_1 \leq k \leq k_2\})$ corresponding to the solid particles launching sites;

- $\rho_{\Lambda p}$, $integer$: amount of particles released at each time step in the launching areas, which shall be 95% of θ_{frz} (*i.e.* the site is not filled but almost, as if it was filled up to θ_{frz} the cell would solidify, thus modify the original landscape);

- ϵ_p, $real$: the factor to improve the particle transport efficiency (it should be approximatively $1/max(\mathbf{u})$, see section 3.1.2);

- ζ_p, $real$: probability for a deposited particle to get eroded, possibly multiplied the non-equilibrium momentum flux tensor (see section 3.1.4).

Some procedures are also stored in this module:

- `part_init`, setting up the above variables;

- `part_transport`, reshaping the particle distribution based on the local fluid velocity and gravity (section 3.1.2), and freezing the particles pointing towards a solidified cell (section 3.1.3);

- `part_erod`, eroding particles according to the algorithm presented in section 3.1.4, and calling the toppling mechanism (with some subtility, as toppled particles should not be reinserted in the aeolian phase transported by the fluid flow) whenever necessary;

- `part_launch`, launching $\rho_{\Lambda p}$ particles from the sites contained in Λ_p elements;

- `part_stream`, streaming the particle densities along the lattice edges.

Solid shape information module (*solid*)

Each cell must know the information about whether it is solid or not, and which cells are solid ones around. This is of critical importance to avoid stremaing a flying particle inside a solid cell, or to make the fluid react correctly (bounce back conditions). It may also be useful to record if a given site was solid because it is part of the initial landscape shape, or it has only grown to be solid under the flying particles freezing process.

There is obviously several ways of dealing with this representation. The first one would be to store *logical* in the same manner as solid particles are stored in the previous paragraph. However, declaring $q+2$ logical fields, *i.e.* $q+2$ is squandering, as each *logical* holds 1 memory word.

As each information can be handled by a single bit, it may become more rational to store the whole information on a single word and to declare the array s_{map}, $integer(n_z, n_y, n_x)$. For each lattice node the information is store in the following manner:

- bit 0 is one $\Leftrightarrow$ the site is solid;
- bit i is one, $i \in \{1 \ldots q\}$ $\Leftrightarrow$ next site in direction $i-1$ is solid;
- bit $q+1$ is one $\Leftrightarrow$ the site is solid from the initial configuration; this information is pertinent when for example, the user wishes to delete all the deposited particle to restore the initial conditions;
- remaining bits can be used to store a short integer indicating the distance to the nearest solid cell (this information have been used in some modification of the C_{smago} constant nearby the ground, in the 2D model).

Several subroutines are called to manage these informations:

- `Solid_stream`, streams the information: the bit indicating if a cell is solid (bit 0) is streamed to neighbor cells to build their local s_{map} configuration map; it can be also useful to store in the end bits the distance to the closer solid site (this distance is taken as the minimum one from any of the surrounding cells $+1$, so the value is dynamically updated when the solid configuration evolves);
- and several procedures to create the initial landscape, drawing simple geometrical object (cylinder, plate, hexahedron ...). Among these subroutines, the most useful ones from a practical point of view are the one which spread a 2D complex profile along the third dimension, and the one which takes as input a xy matrix where each element represents the altitude of the ground (for the latter one, it may be useful to enter the landscape information as a set of p given points $\{x_i, y_i, z_i = z(x_i, y_i)\}_{i=1 \ldots p}$; then a Delaunay triangulation can be computed and the altitude for a point inside a defined triangle is straightforwardly interpolated).

Rendering modules

To render the previous informations, *fluid_render, part_render* and *solid_render* modules are designed. They contains procedures to save the information in several manners for a later treatment: profiles (fluid velocity/vorticity, particle deposits) to be plot with *gnuplot* or *xmgr*, and *Avs/express* fields or UCD files (see appendix C) for a 3D visualization.

Pilote device (*cockpit*)

To offer an efficient way of handling a program with many variables and features, it is interesting to design both an interactive and a batch command interface (see appendix B for more details). These procedures are grouped within the *cockpit* module.

4.2.3 Modules organization

Altough the *module philosophy* is not a recent matter for computer scientist (it appeared in the eighties and had its hour of glory with *modula 2*), it is rich enough to build a structured program as in the one we had in this case. Even if it is far from reaching all the possibilities offered by object oriented languages (*Java* or *C++*), it is of a much simpler use in *Fortran 90/95.*

A module contains types, variables and procedures (functions or subroutines) either private or public. Any entity (the main program, a procedure or a module) *using* a given module has access to all its public features. Moreover, all the entities using this module will share the values of its variables, thus ensuring consistency.

Therefore, a large program (at least up to a certain size), can be organized as a graph (oriented by the dependence and obviously non-cyclic) of modules. Such a graph for our application is displayed in figure 4.2.

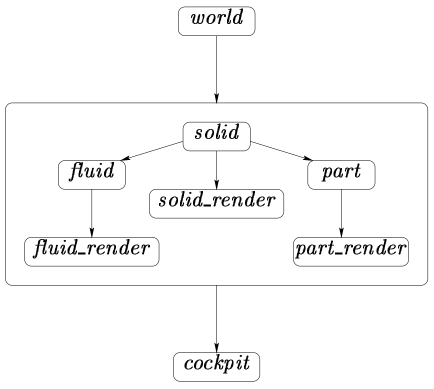
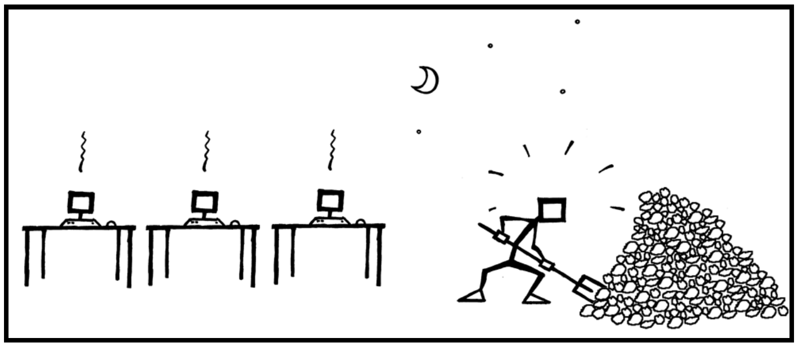
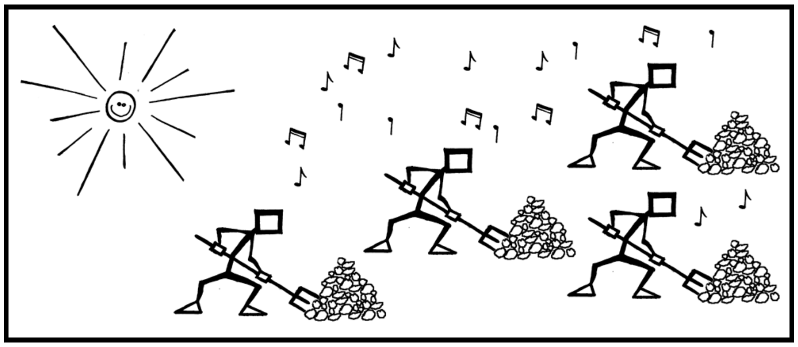


Figure 4.2: Module dependency graph; an arrow pointing from *solid* towards *part* means that the module *part* uses *solid*; every module uses *world* and all of them are used by *cockpit*. This graph can be directly translated in a *makefile* if each of those modules are contained in separated files.



OR



Sharing efficiently the work load among ressources: the essence of parallelism

4.3 Parallelizing the code

When time comes to run large simulations, *i.e.* memory and time consuming problems, the user's patience can be severly tested. Several solutions can be considered:

- go away and come back a week later to see the results;
- buy a new very powerful (and very expensive) machine to run the same code in a shorter time;
- share the work among several process units linked together with an efficient connexion network.

The first solution is often chosen in many scientist's everydays life (and usually when this scientist is not from a computer science background but has important motivations in using numerical model for his research). When his budget is a bit comfortable, the second solution can be adopted; however, the race for the fastest machine soon becomes very expensive and performance is bounded by technology.

However, if he needs fast computing times (for prediction purposes or when too long simulations are incompatible with model exploration), the programmer can decide to choose the third solution. Thus, to conquer the results, he can divide the problem among a set of machines and therefore *parallelize* his code.

4.3.1 Parallel programming models

Several parallel programming models dedicated to high performance computing currently exist. Although the aim is always to share the working load among different computing units (or nodes, PE), this division can be undertaken in several ways:

- **Data parallel, SIMD** (Simple Instruction Multiple Data): every node contains a part of the data, and all of them execute synchronously the same instruction (either computations or communications) with their own data. This programming model had its success but was proven to be efficient only for a limited range of problems and no more general purpose computers are built using this model. Meanwhile, the programming philosophy (one instruction executed throughout a whole array) has been kept in *Fortran 90*, and is even implemented on clusters of machine in a totally transparent way (but often with disappointing performances) with *HPF* (High Performance Fortran).

- **Shared Memory**: a set of processors work on the same problem (each one executing its own code), but the information is stored in a shared memory system, and therefore can be accessed by anyone at any time (to ensure data consistency, some locking procedures must be implemented). Although the shared memory programming model is intellectually very attractive (no explicit communications), it is very dependant on the machine architecture, hardly scalable (excepted on the Sun Entreprise 10000) and often quite expensive.

- **Message Passing**: each process, associated with a process identifier, executes a copy of the main program (or even different programs) and communicate with the other nodes sending and receiving explicitely messages.

Once again, comparing extensively these three methods is far beyond the purpose of the current work (see [Kumar *et al.* 1994] for a more complete discussion). However, we may argue why we have chosen the Message Passing model for the present program.

The message passing model can address a much wider range of problems than the first two, but cellular automata models (local computations and synchronous nearest neighbors communications) can be handled very efficiently and very elegantly by any of them. We have in fact implemented the 2D version of our program on a *Connexion Machine CM-200* (by *Thinking Machine*) using the archetype of a SIMD architecture.

However, the message passing model is very portable and is seen as a high level, language independent, communication library upon any set of standard machines linked either with a standard TCP/IP or a dedicated network. Moreover, even if commercial products exist (often provided with dedicated communication devices), message passing libraries are provided for free on almost every operating system.

In the past few years, two message passing standards have emerged:

- **PVM** (Parallel Virtual Machine): can be ran among a heterogeneous set of machines, with dynamic process managment.
- **MPI** (Message Passing Interface): to be ran mainly on a homogeneous set of computers, with a much richer set of communications procedures (either point to point or collective).

Even if these features seem rather fundamental, the kernel of both libraries are only *send* and *receive* procedures. Moreover, they both converge towards the same "ultimate" message passing model (a new version of a library is mainly achieved by picking up the best features of the other one), and *MPI 2* (which is currently only developed for a Hitachi architecture) seems to take the advantage.

We have choosen *MPI* to be the library in the current work, partly for the richness of the available primitives, partly for the fact that the software can take advantage of a good interconnecting network.

4.3.2 The Message Passing Interface (MPI) environment

The aim is to distribute the execution of a program over a set of processes (located on different machines). Each of those tasks holds a copy of the initial program (with its own variables), plus some local MPI information which can be gathered during the MPI initialization stage:

- *mp_size*, *integer*: the number of launched tasks;
- *mp_rank*, *integer*: the rank of the calling process in the set, *i.e.* a number between 0 and *mp_size*-1.

If the processes are to be organized (from a logical point of view) as a ring, it can be useful to define the labels of the right and the left hand neighbors:

- *mp_right*=*mp_rank*+1 modulo *mp_size*,
- *mp_left*=*mp_rank*−1+*mp_size* modulo *mp_size*.

4.3.3 Some basic communication procedures

The communications procedures are presented in a simplified form, as standard MPI ones requires several more parameters.

Point to point communications

As stated earlier, the kernel of MPI consists in send and receive procedures, called mainly with the following parameters:

- `send`*(buffer, size, type, dest, tag)*: the task invoking this procedure sends *buffer* (*size* elements of type *type* - integer, real, user defined) to task number *dest*; the message is labeled with *tag*;
- `mpi_recv`*(buffer, size, type, source, tag)*: in the same manner, the task calling this procedure receives (it therefore had to be previously sent) the *buffer* in a message labeled by *tag*.

A message therefore is built from data, source, destination, and a tag. This tag can be used for message identification, but can also be always set to some value if no verification is needed.

The most important idea in message passing programmation is a good organization of the communications: a message sent must be received; if a receive procedure is posted, a corresponding message must have been sent, as shown in the following example.

A simple example

Every task holds a real variable a, and wants to store in a variable a_l the value of a held by the left-hand neighboring process (the one with the precedent rank). Once the initialization has been achieved, the communication part is simply these two lines:

`call send`$(a, 1, real, mp\_right, 0)$
`call recv`$(a_l, 1, mp\_left, 0)$

where the communication tag is 0.

Non-blocking communications

Of course, message passing can be costly: during the communication, the process waits (software and hardware overhead, synchronization with the other process invoved in the communication) in a non-constructive way. Indeed, during these overheads, the CPU could be used to compute data independent from those involved in the transfer. To overcome this problem, non-blocking send and receive can be invoked:

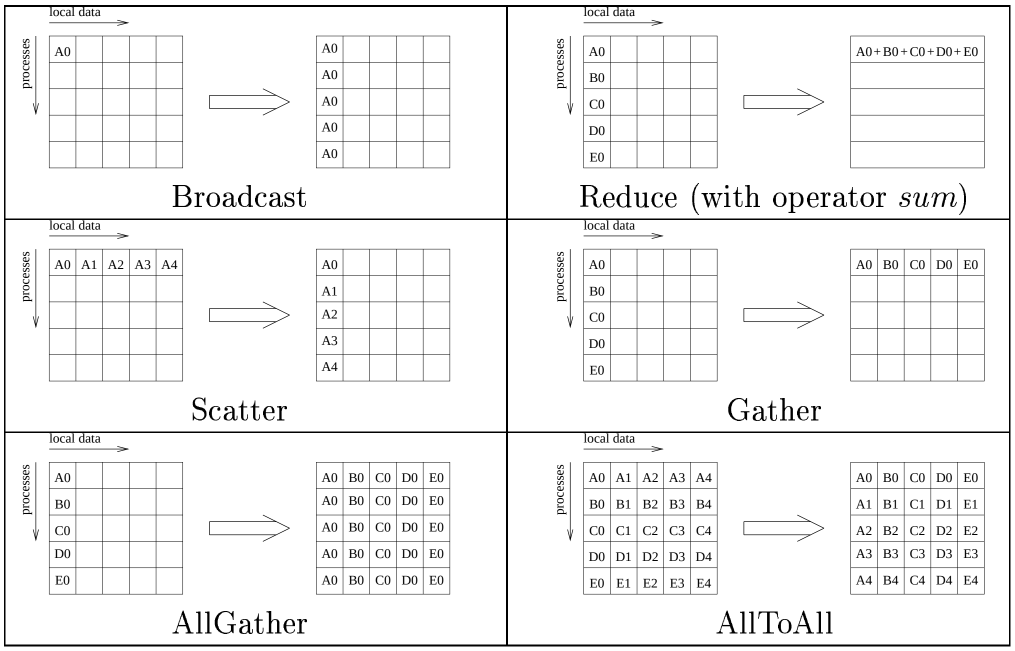


Figure 4.3: six MPI global communication primitives. For each of those, the data distribution (spread among the local memory (horizontally) and the processors (vertically)) is displayed before and after the call in the send and receiving buffers.

- `mpi_isend`*(buffer, size, mpi_type, dest, tag, req)*: where `i` states for immediate (non-blocking),
- `mpi_irecv`*(buffer, size, mpi_type, source, tag, req)*,
- `wait`(*req*).

Once `mpi_isend` or `mpi_irecv` is called, the procedure exits without waiting for the communication completion (which is achieved in background) and a request id *req* is returned. However, a call to the `wait` procedure will force the program to stop until the message pointed by *req* is achieved. An example of this feature is given in section 4.3.5.

Although these communications are often more efficient, one should take some precautions using them: as a send is not effectively completed when steping out of the procedure, the *buffer* is not already sent; therefore, modifying it right after the call may, in some situations, send corrupted data.

Global communications

Communications involving all the processes can also be useful as, for examples (see figure 4.3):

- *broadcast* of an information held by one process to all the others,
- *gather* of informations stored in all the processes onto one,
- *reduce* with some operator (such as *sum*, *max*, *product*) of informations stored in all the processes onto one,
- and other more general variants.

But MPI is a lot more

If send and receive primitives are the kernel of the library together with collective communications, there are many other features which can be used in efficient and elegant progamming, such as:

- initialized communications: a message (buffer, tag, source or destination) is prepared at once and associated to a request; invoking only this request will later start a commnuication with the data stored in the buffer at that moment; this method reduces a lot the software communication overhead;
- *buffered*, *ready* and *synchronous* communications offer different way of managing the buffers, allowing a more efficient handling of large messages: instead of relying blindly on the local MPI implementation to manage the buffering of messages, the programmer can force the message to be fully buffered, to exit the send procedure once a corresponding receive has been posted or when the receiving process has started (such subtility are used for the manging of large messages, when the pre-defined buffers are not sufficient);
- *wildcards*: to receive message with unknown source or tags;
- *inquiry* primitives: to get informations about a message (is it yet arrived? tag? source? type?);
- *topology*: processors can be arranged in subsets, and some global communications can be called only within these subsets;
- *parallel I/O* (as I'O must currently be done sequentially, thus are a bottleneck for many applications), *one sided communications* (instead of posting a `receive` to any corresponding `send`, a task can directly pickup data contained by another one), and some more, are new features of MPI-2.

4.3.4 Sharing (distributed) array with MPI

As stated earlier, each process has a copy of the program. If the aim is to manage a big array, we can suppose that every task knows its size $n \times m$ (if only one task has read these parameters, they can be broadcasted at once).

The idea of parallel processing is to divide the work, each process will only allocate a part of the initial array. This share can be done in several manner:

- in strips, the easiest way;
- in rectangles, to lower the ratio between boundary length and domain size; this method has been implemented in 2D with dynamic load balancing (*i.e.* the rectangle domains are continuously adjusted to equilibrate the computation time between all the processes) in a very efficient way by [Martin *et* chopard];
- in totally irregular shares by [Dupuis *et* Chopard 1999] (2D), where void cells (inside solid for example) are not even allocated; however, this technique involves irregular boundaries and expensive communication and allocation processes.

For our problem we have chosen the strip method as it is the simplest but also efficient enough: the main drawback, *i.e.* the ratio between boundary length and domain size vanishes a bit in 3D, where communication overheads can be overlapped by local computations as it will be shown later.

Therefore, if for example a 2D array of size $n \times m$ is to be shared among mp_size tasks, each of those tasks has to allocate a local array of size $n \times m_{loc}$, where $m_{loc} \sim m/mp\_size$.

Nevertheless, as mp_size does not always divide m, it is more precise to define:

```
if($mp\_rank$<mod($m$,$mp\_size$))then
   $m_{loc}$=$m$/$mp\_size$+1
   $m_{offset}$=$mp\_rank$*($m$/$mp\_size$+1)
else
   $m_{loc}$=$m$/$mp\_size$
   $m_{offset}$=$mp\_rank$*($m$/$mp\_size$)+mod($m$,$mp\_size$)
end if
```

where m_{offset} is a local variable storing the position offset of the slice allocated on the current process within the global array; in fact, this information is not compulsory for many computations such as a classic CA, but is needed when irregular information is to be spread onto the whole array (such as a map read in a file, where each process has to take only one slice of the data).

Moreover, the distribution shown here (and in figure 4.4) is balanced on homogeneous PE, but it could be done in other manners depending on the situation:

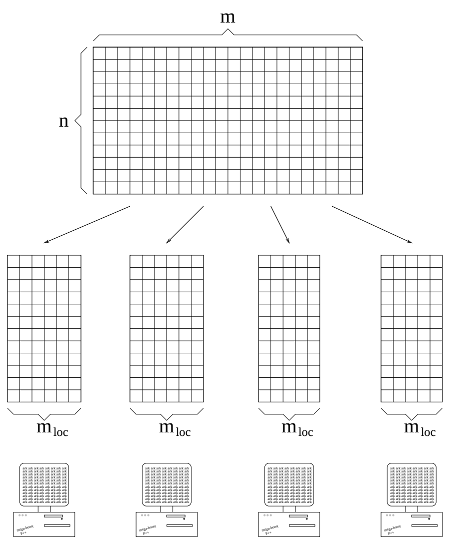


Figure 4.4: distributing an array of $n \times m$ ($m = 22$) among a set of $mp\_size = 4$ processors. Each node has an array of $n \times m_{loc}$, where $m_{loc} = 6$ for the two first tasks ($mp\_rank < (m \mod mp\_size)$) and $m_{loc} = 5$ for the two last ones.

- if the tasks 1 to 3 are known to run on CPU twice faster as the remaing ones, it may be clever to allocate more data one them;
- in the case of a *master/slave* design, one task can be reserved for other work than parallel array computations, therefore the allocation should be made only on the remaining nodes.

Once each processor has its own array (in fact a share of the global one), it can execute whatever computations on its data. However, if there is a need for data stored on another processor (*i.e.* elsewhere on the global array), communications must be foreseen. The example of lattice models is described in more details in the following subsection.

4.3.5 Lattice gas models and MPI

From their basic numerical scheme, CA, and more generally lattice gas models, are ideally parallel as a time step consists in two stages:

- collisions, *i.e.* computations based on local variables;
- streaming, *i.e.* communications with the nearest neigbors.

As the domain is represented by a regular lattice, thus an array, it can be distributed as shown in the previous subsection. The first stage is naturally achieved as each node computes data contained in its array. In the second stage, communications inside the array (for inner columns according to figure 4.4) does not pose any problem. However, border cells (outermost columns in figure 4.4) require communications with the nearest processors.

Therefore, a computing step can be implemented as:

```
1. local computations according to the evolution rule
2. local communications with the nearest neighbors

3. send information to the right PE
4. send information to the left PE
5. receive from left
6. receive from right
```

However, during the communication overhead, the CPU is unemployed (thus some power wasted). To solve this problem, it is possible to overlap inter-processors communications and computations [Luthi 1998]:

```
1. non-blocking send information to the right PE
2. non-blocking send information to the left PE
3. non-blocking receive from left
4. non-blocking receive from right

5. local computations in the inner domain

6. local communications in the inner domain

7. wait the completion of both receive
8. local computations on the outermost cells
```

From a practical point of view, it often not very easy to overlap communications and the whole evolution process, as there are treated at different location (once more, an object approach could be more flexible on that point). However, it has been measured to be sufficient to overlap external communications with inner ones, especially when the connecting network is fast enough .

4.3.6 Among which dimension sharing the array?

Problems addressed by lattice model concern 2D or 3D array ($n_x \times n_y$ or $n_x \times n_y \times n_z$). The question of which of the domain dimensions should be split is not obvious and two factors must be taken into consideration.

Sharing along the largest dimension

The dimension among which the array is split, as described in section 4.3.4, should be the largest for two reasons:

- the imbalance between shares with $+/-$ one slice will be less important,
- the size of communication buffers (the product of the remaining dimensions) will be the smallest, optimizing the message transfer time, proportional to the buffer size.

As practical applications of our model can be seen as a wind tunnel experiments, it is natural to consider the $x-$ dimension (the one along the flow stream) as the largest.

Putting data in memory

However, a question remains about the declaration of an array a of size $n_x \times n_y$: should it be declared as $a(n_x, n_y)$ or $a(n_y, n_x)$.

This question is language dependent and is based on the element order in memory. In Fortran, contrary to C, the first indexes varies the most quickly, *i.e.* an array $a(2,3)$ is stored in memory (linearly of course) as:

a(1,1)	a(2,1)	a(1,2)	a(2,2)	a(1,3)	a(2,3)

To store the array in a way that subsets sent in messages are in contiguous memory location, it is clever to distribute the array among the last dimension. Thus, one should declare $a(n_y, n_x)$.

These explanations are consistent with data declarations presented in section 4.2.2, where it seemed at the first glance not evident to declare the fluid array as $real(-1 : q_f - 1, n_z, n_y, n_x)$ instead of $real(n_x, n_y, n_z, -1 : q_f - 1)$.

4.3.7 Load balancing

Working with a large set of computers confront the programmer to untraditionnal problems: the load of these machines can be unbalanced (either because other users are logged, or because some sub-domains require less computational effort: if there is a lot of solid cells for example).

The load balancing presented in section 4.3.4 is static and the most regular possible. Far away from sophisticated methods consisting in managing time dependent rectangular blocks [Martin *et* chopard], an simple algorithm can be stated:

- time to compute a domain slice is measured (only the computations, not the communications as they depend on the other tasks),

- neighboring processes exchanged their effective computational times,
- if a process is really fastest, it can take a slice from a slower neighbor.

Although this method is rather simple, some precautions must be taken:

- time measurements should be done over several evolution steps, as the load balancing process is expensive;
- a slice removal should be achieved according to a probabilistic rule, slowing down the convergence to the most optimal ditribution, but lowering the risk of oscillations;
- all data must be carefully moved, to ensure the model consistency.

Such a load balancing algorithm has been implemented with an early version of our $3D$ program and proven to be efficient (a processor loaded with other time consuming jobs was jettisoning domain slices to its neighbors. However, as most of the computations were achieved on parallel cluster of machines in single user mode, the priority was transfered on the implementation of the model itself and loadbalancing module abandoned.

4.3.8 MPI implementations used

As Fortran 90 and MPI are recognized on almost all the platforms, our program could be compiled and ran on several pararlel machines:

- 32 500Mhz Pentium, with the free *MPICH* library;
- 8 Sun Ultra 5, interconnected with the *Scali* sofware and hardware;
- 14 nodes IBM SP2, based on RS6000 processors, with a native MPI implementation;
- 12 nodes Sun Entreprise 10000, which is a shared memory machine, but can be used with a native message passing library.

Although each of those machines have been extensively used, we have made the largest part of the production simultations on the Pentium cluster. A benchmark, to compared MPI performance on different platforms, is presented in appendix D.

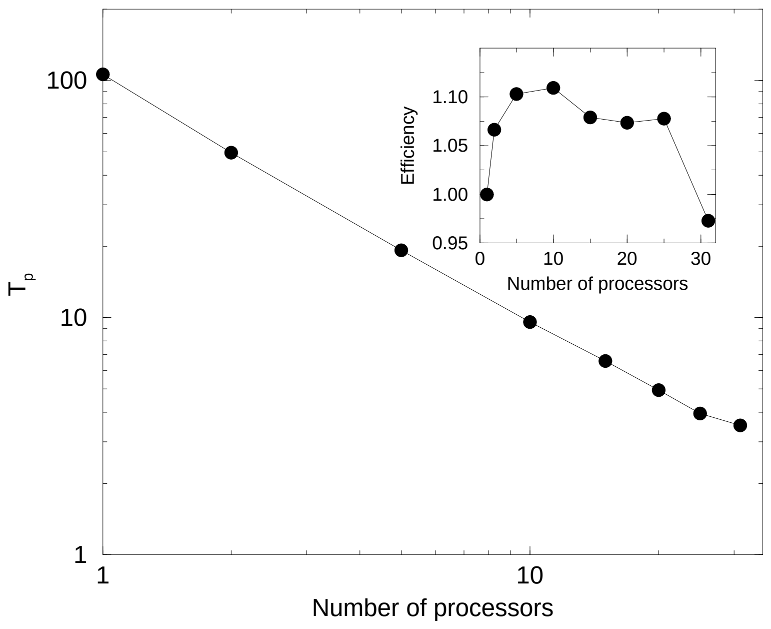


Figure 4.5: scaling performances on a Pentium III cluster: the parallel time is plotted versus the number of processors on which the program was run (domain size is $200 \times 3 \times 10$ and the time is plotted for 500 iterations). The inner graphe show the efficiency, *i.e.* the sequential time / (parallel time $\times$ number of processor): a value of 1 indicates a perfectly scalable application. However, an efficiency greater then 1 shows that our application is even superscalar (a better gestion of cache memory is often the cause).

4.3.9 Is parallelization efficient?

It was claimed in the beginning of this section that the purpose of parallelism is to get shorter computation times. In a perfect world, a program would run twice faster on a two-node parallel machine than on a single processor (five time faster on a five CPU one and so on).

Figure 4.5 shows how our application, on the Pentium III cluster, evolves. The application is proven to be super scalable, *i.e.* it runs more than p times faster on p processes than on a single node. This super-scalability can be explained, as sharing a large array among p processes reduces more or less by a factor p the allocated memory on each PE. Therefore, all the data can fit inside the memory cache and remain there, lowering the access time to it.

4.3.10 Summary

In this chapter, we have presented the language we have used, Fortran 90, how we have applied its module philosophy to our problem, and how we have parallelized the code.

Although it is disclaimed par computer scientists, they often forget that Fortran 90/95 has been designed to achieve fast computations on a restraint range of applications (array computations, mathematics applications). It does not have the "all-purpose" ambition (and often the over-weight) claimed by object oriented languages (mainly C++ for high performance computing).

Finally, it has appeared to be a very efficient and easy to manage programmation language for our application. However, some limits were found, when we thought about elegant style for programming load balancing for example, but the global programming of a complex lattice gas model is very well suited to Fortran 90.

To parallelize the code, we have first ran a $2D$ code on a data parallel machine (the Connection Machine CM-200). However, the $3D$ code was programmed on cluster of nodes linked with a message passing library, namely MPI. The genericity of Fortran 90 and the MPI library has allowed us to compile and run the very same code on various architectures: IBM SP2 (14 RS/6000 processors), an eight Ultra 5 cluster, and a 32 Pentium cluster. Parallelization was naturally suited to the lattice gas model (the regular domain is shared in slices) and computations times were super scalar (running on 30 node was more than 30 times faster than running on a single node).

This experience show how the efficient parallelization of a code can change the scientist's life. For example, reducing on our PC cluster by a factor 30 simulations times has opened the door to fast simulations (most of the results presented in this work did not last more than 15 minutes), and therefore to really interactive investigations.

Chapter 5

Conclusion

The mesoscopic level of description, as described in this work, offers a totally different approach to complex system simulations and has been applied to a large panel of applications. This approach was therefore a natural one to model snow transport by wind, as a description in term of microscopic rules is a very attractive alternative to classical models based on the resolution of semi-empirical equations. Instead of extracting complex governing equations and later modeling them using complicated numerical schemes, it appears much easier to incorporate directly the phenomenon ingredients in intuitive evolution rules. For example, according to some authors, the ejection of particles might be provocated by the landing of another one, directly or after a short delay: including this rule at a macroscopic level can be hard to manage, whereas putting it in a microscopic description of the model in trivial. As stated by Feynamn, the very details of microscopic interactions are not important for the global system behavior, as soon as the mesoscopic rule catch the main components.

In this work, modelling both a fluid and solid particles have shown how complex behaviors can be caught, even if the fictitious world we have built seems too simplistic:

- a non-steady turbulent flow was produced by a simple subgrid BGK model;
- this fluid is naturally defined for time evolving solid boundaries and recovers many features found in the literature;
- snow deposits range over very different scales, from small ripples to full Alpine crests, with very good agreement with outdoor results;
- eventhough different particle transport modes (creeping, saltation and suspension) are not explicitely defined, the model naturally produces them;
- the very same approach have been used successfully for sand underwater transport with very accurate results.

Moreover, the parallel nature of lattice models makes their implementation very easy (starting from scratch, a $2D$ fluid solver takes a couple of hours of programmation) and very well suited to parallel computations. Instead of waiting for days, a standard computations presented in this works takes between 15 minutes and a couple of hours on a 32 Pentium III cluster, a very inexpensive parallel machine.

This model elegance, together with short computation times make such a model a very interesting and comfortable simulation tool, with which testing new scenarios, investigating new leads, does not catch the scientist into despairing situations.

However, there is another side to the coin. As we have seen thoughout this work, building a ficticious world induces the loss of an explicit spatial scale and the use of parameters hardly connectable to field observation. In the particle models, natural questions can arise: what is the size of a modelled snow flake? how coherently can ζ_p (the particle erosion probability) be related to reality?

As the only validation of a model cannot come only from results it produces, its rules must be explicitely described and argued. This disconnection with the real world may sometimes be seen as a weakness by some persons.

Nonetheless, we believe our model has caught interesting features, and has taken up the challenge. However, much work can still be achieved, with the aim, for example, to develop a more powerful forecasting tool (to build works, such as fences or slanted screens, in complex situations).

Despite trying to incorporate more of the complex components coming up in the blowing snow process (*e.g.* cristal modification, deposit evolution and constraint), it may be more effective to concentrate at first on different lattice model improvments:

- continuous (or off-lattice) solid definition; instead of having step defined ground configurations, some investigations should be undertaken on a smoother definition; this might be very important on all deposit with gentle slopes (such as deposit around fences, some crests) [Chen *et al.* 1998];

- better subgrid model, not depending on a global and empirical parameter such as C_{smago}[Chen 1999];

- non-uniform scale or unstructured lattice, with smaller cell size at ground level for instance [Peng *et al.* 1999].

Appendix A

A multiparticle fluid model

A.1 In between CA and BGK models

Lattice gas automata (section 2.1.1) and BGK models (section 2.2) offers the same kind of approach, dealing with particle travelling on a regular lattice, with synchronous, relatively simple, and local collision rule. However, more than only their discrete/continuous representation of the particles, they differ from several points of view:

	LGA	**BGK**
Representation	discrete	continuous
Collision term complexity	exponential with the number of particles (lattice links)	linear with the number of particle and the dimension
Statistical noise	very high	low (initial conditions or external noise)
Viscosity	determined by the collision rule and the density (very narrow range)	tuned by an external parameter (can reach very low values modulo the subgrid model)
Navier Stokes	modulo a slight Galilean invariance and a spurious pressure term	OK
numerical stability	ensured by the exact computations	low viscosity makes the system to blow up
N-body correlations	taken into account (particle are indentified)	vanish with the Boltzmann molecular chaos hypothesis

For the major problems addressed by fluid simulation, correlations between particles can be neglected and lattice Boltzamnn models are satisfying. However, for some questions, for example the *ballistic annihilation problem* (particle are annihilated when they encounter) or *reaction-diffusion systems* (chimical reaction where particle encounters are crucial) [Cornell *et al.* 1991], the density fluctuations are crucial.

Meanwhile, those questions cannot be approached with LGA automata model, as intrinsic spurious invariant due to the simplicity of the collision rule distort the result. Therefore, the classical numerical approach is an extensive molecular dynamics, computing exact position and intereaction between the particles, very expensive from a computational point of view.

We will propose in this appendix a mix between LGA and BGK models, *i.e.* a model where particles are discrete, but where the maximum number of particles per cell is not fixed and which obeys BGK inspired collision rules.

In the following section, we will adapt our model to the *2D ballistic annihilation* problem and focus on the the decay law.

A.2 The multiparticle model

Our multiparticle microdynamics is described in terms of quantities $F_i(\mathbf{r},t) \in \{0,1,2,...,\infty\}$, giving the number of particles entering site $\mathbf{r}$ at time t along lattice direction i. Following the standard CA or LB approaches, the evolution rule of our model reads

$$F_i(\mathbf{r}+\mathbf{c}_i, t+1) = F_i(\mathbf{r},t) + \Omega_i(F(\mathbf{r},t)) \tag{A.1}$$

where Ω_i the collision term.

The key idea of our model is to write a collision rule for the integer quantities $F_i(\mathbf{r},t)$ so that, on average, they follow the standard BGK dynamics (see section 2.2. We assume that the main effect of the interaction Ω_i is to restore a local equilibrium distribution f_i^{eq} along each lattice direction:

$$f_i^{eq} = \rho t_p \left[1 + \frac{\mathbf{c}_{i\alpha}\mathbf{u}_\alpha}{\mathbf{c_s}^2} + \frac{1}{2}\left(\frac{\mathbf{c}_{i\alpha}\mathbf{u}_\alpha}{\mathbf{c_s}^2}\right)^2 - \frac{\mathbf{u}_\alpha\mathbf{u}_\alpha}{2\mathbf{c_s}^2} \right]$$

where $\rho = \sum_i F_i$ and $\rho\mathbf{u} = \sum_i F_i\mathbf{c}_i$. The above expression is the standard form of the local equilibrium already desribed in equation 2.7. Note that F_i's are integers whereas f_i^{eq}'s are real numbers.

The relaxation of F_i to the local equilibrium f_i^{eq} is governed by a parameter τ. Thus, like in the LB case, we require that the number of particles F_i' leaving, after collision, a given site along direction i be given by

$$F_i' = \frac{1}{\tau} f_i^{eq} + \left(1 - \frac{1}{\tau}\right) F_i + \Delta F_i \tag{A.2}$$

where ΔF_i is a random quantity accounting for the fact that, after collision, the actual particle distribution may depart from its ideal value, due to the integer nature of F_i'.

Algorithmically, F_i' is computed as follows. Let $\rho = \sum_i F_i$ be the total number of particle at the given site. We assign to each direction i a weight w_i

$$w_i = \max\left(0, \frac{1}{\tau} f_i^{eq} + \left(1 - \frac{1}{\tau}\right) F_i\right) \tag{A.3}$$

From these weights, we define p_i, the probability for a particle to leave the site along direction i, as $p_i = w_i/M$, where $M = \sum_i w_i$ is a normalization constant. When $w_i > 0$ for all i, $\rho = M$. However, the right-hand side of equ. (A.2) may become negative. This leads to numerical instabilities in standard BGK models.

To compute the collision output, we run through each of the ρ particles and place them in direction i with probability p_i. This gives us a temporary particle distribution $\tilde{F}_i$ which satisfy (when $M = \rho$)

$$< \tilde{F}_i >= \frac{1}{\tau} f_i^{eq} + \left(1 - \frac{1}{\tau}\right) F_i \tag{A.4}$$

For large enough ρ, the particle distribution can be computed with a Gaussian approximation[Chopard *et al.* 1994]. In this way, the algorithmic complexity of the operation does not increase when ρ becomes large.

While the distribution $\tilde{F}_i$ of outgoing particles obviously conserves the number of particles, equation (A.4) shows that it does only conserve momentum on average and some particles must be redirected to ensure exact local conservation. The actual number of particles, F_i', leaving the site after the multiparticle collision is obtained from $\tilde{F}_i$ through a trial-and-error procedure which reorganizes particles among the directions until the correct momentum is obtained.[Chopard *et al.* 1998b]. From the way the particles are distributed, we expect that roughly $\sqrt{\rho}$ of them are misplaced. This gives an estimate of the number of iterations requested to re-adjust the particles distribution.

According to the above discussion, the quantity ΔF_i defined in equation (A.2) vanishes on average. This fact is confirmed numerically.

Consequently, with $F_i(\mathbf{r} + \mathbf{c}_i, t + 1) = F_i'(\mathbf{r}, t)$, we write

$$F_i(\mathbf{r} + \mathbf{c}_i, t + 1) \approx \frac{1}{\tau} f_i^{eq}(\mathbf{r}, t) + \left(1 - \frac{1}{\tau}\right) F_i(\mathbf{r}, t) \tag{A.5}$$

Equation (A.5) is identical to the usual BGK microdynamics[Qian *et al.* 1996b], except that now it approximates a multiparticle dynamics in which F_i are integer quantities. Therefore, we expect the same hydrodynamical behavior to emerge: equation (A.5) is equivalent to the Navier-Stokes equation with viscosity $\nu \propto (\tau - 1/2)$.

However, the fact that ΔF_i is only zero on average, produces an extra contribution to ν. It is known from kinetic theory that viscosity is proportional to the particles mean-free path. In the multiparticle interaction, an amount ΔF_i of particles will be misplaced and this is equivalent to an effective collision process, which reduces the mean-free path and modifies the above expression for the viscosity. The next section quantifies this effect.

The present muliparticle scheme is intrinsically stable. No small fluctuation will be amplified unphysically to make the arithmetic blow up when $\nu \to 0$, as happens with the LB model. Any value of the relaxation parameter τ can be considered without numerical problems. The behavior of our model when $\tau \to 1/2$ is briefly studied in the next section.

A.3 Flow simulation and viscosity

This section presents the result of flow simulations obtained with our multiparticle dynamics. A first test is to reproduce a Poiseuille flow in a channel. We assume that the x coordinate is along the channel length. No-slip (bounce back) boundary conditions are imposed at $y = \pm a$, where $2a$ is the channel width. Fig. A.1 shows the velocity profile measured in a laminar flow regime. We observe the expected parabolic velocity profile[Tritton 1988]

$$< u_x(y) >= \frac{G}{2\rho\nu}(a^2 - y^2) \tag{A.6}$$

where G is the external force per lattice site (or the pressure gradient). In our case, G is produced by moving with probability p_G each particle from lattice direction i such that $\mathbf{c}_{ix} < 0$, to direction i'', where i'' is the direction symmetrical to i with respect to the y-axis. We have $G \propto \rho p_G$.

From equation (A.6) one can obtain the viscosity ν as

$$\nu = \frac{G}{2\rho u_{max}} \tag{A.7}$$

where u_{max} is the velocity at the middle of the channel, i.e. where $y = 0$. The viscosity is assumed to vary as $\tau - (1/2)$. Thus, by varying τ, we should observe a corresponding variation of u_{max}. However, if the viscosity becomes small enough, we may reach a turbulent regime with a different velocity profile[Tritton 1988]. To avoid this difficulty, we vary G proportionaly to $(\tau - 1/2)$, which should yield a constant u_{max} value.

Fig. A.2 gives the results for u_{max} obtained for various values of τ. There is a region where u_{max} is constant, as expected. However, when $\tau < 0.65$, u_{max} decreases, as an indication that the actual value of ν departs from the $\tau - 1/2$ behavior and does no longer decreases. Thus, there is a minimal built-in visosity in the model. From the numerical point of view, a value τ close to $1/2$ may

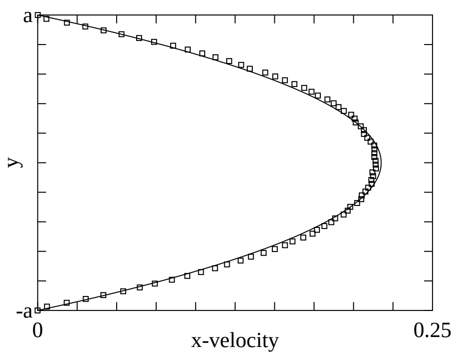


Figure A.1: Velocity profile in a multiparticle Poiseuille flow. The plot shows the horizontal average velocity $< u_x(y) >$ as a function of y the vertical position between the upper and lower boundaries. The solid line corresponds to the best parabola fit.

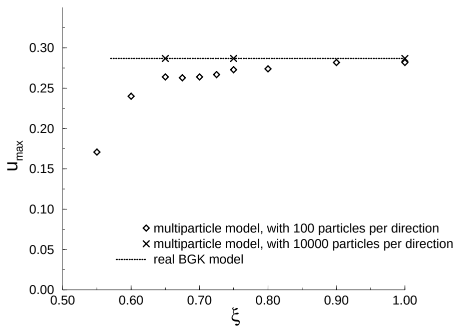


Figure A.2: Measured value of u_{max} as a function of τ for $G \propto [\tau - (1/2)]$. The classic BGK result fits exactly the theoritical prediction, where $\tau \geq 0.57$ (for $\tau < 0.57$, numerical instabilities occur). The integer particles model is stable, but when τ decrease, probabilities p_i may become negative; forcing them to 0 (see equ. (A.3)) increases the effective relaxation time. Note that the more particles there are, the closer the simulation is to BGK values.

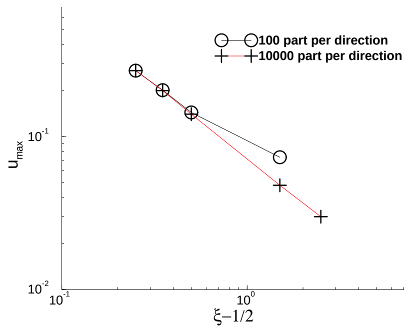


Figure A.3: Measured value of u_{max} as a function of τ for G constant. According to (A.7), $\frac{\tau-1/2}{u_{max}}$ should be constant. This is the case for simulations with 100 and 10000 particles per direction. However the range of validity is larger as more particles are present. This is explained by the emerging importance of the term ΔF_i in equation (A.2), varying as $\sqrt{\rho}$.

cause the probability p_i for a particle to leave the site along direction i to become negative. In this case, $\tilde{F}_i$ is zero and only the procedure of momentum tuning will populate this direction. This results in an effective viscosity. In order to make this point more quantitative, we compare, in figure A.2, the values of u_{max} obtained with both our multiparticle model and a standard D2Q9 lattice BGK model. On the other hand, the stochastic nature of our algorithm does not allow a correct behavior with high relaxation time. The range of correct behavior can be enlarged, see fig. A.3, by using more particles (thus lowering the statistical noise)

Finally, as a last example of flow, we present in fig. A.4 a simulation of a 2D von Karman street.

A.4 Ballistic annihilation

This section presents another application of our multiparticle model: the ballistic annihilation problem. Ballistically controlled reactions provide simple ex-

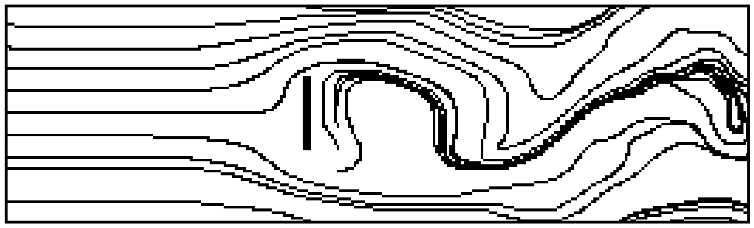


Figure A.4: snapshot, in a multiparticle simulation of a von Karman street.

amples of non-equilibrium systems with complex kinetics[Elskens *et* Frisch 1985, BN *et al.* 1993, Rey *et al.* 1995, Rey *et al.* 1998]. They consist of a system of particles moving freely with given velocities until they experience a collision. When two particles meet, they instantaneously annihilate each other and disappear from the system. In this problem, N-body correlations are expected to play a role on $N(t)$, the particle number decay. The behavior of $N(t)$ is assumed to be described by a power law $N(t) \sim t^{-\alpha}$, in the long time regime.

In one dimension, systems with only two possible velocities $+v$ or $-v$ have been studied by Elskens and Frisch[Elskens *et* Frisch 1985] and yield $\alpha = 1/2$. The case of a general velocity distribution has been treated analytically by Droz *et al.*[Rey *et al.* 1995]. It was shown that different dynamical behaviors can occur depending on the initial velocity distribution and that fluctuations play a very important role, invalidating the predictions of a mean-field approach.

Beyond one dimension, the situation is much more complex. For a continuous space time system, a numerical integration of the Boltzmann equation for the time evolution of the velocity distribution with a uniform initial condition leads to the value $\alpha = 0.91$.[BN *et al.* 1993]

A recent two-dimensional molecular dynamics study (with up to 10^5 particles) by [Trizac] gives a decay exponent α whose value is affected by finite size effects and varies between 0.86 and 0.89 depending on the size of the sample. Moreover, it is observed that the kinetic energy distribution function evolves in time towards a Maxwellian, although the results of the simulations are very noisy.

Due to the way our multiparticle is defined, intrinsic fluctuations are present in the dynamics and ballistic annihilation can be used as a test to check whether the particle correlations are dealt with in a physical way. It is easy to add to the previously defined rules a new mechanism implementing the annihilation of each pair of particles arriving simultaneously at the same site with opposite velocities. More precisely, the annihilation rule we use is the following: $F_i \to \max(F_i - F_{i'}, 0)$, where, i' is the direction opposite to i. Note that an annihilation probability less than 1 can also be implemented with a more complicated rule.[Chopard *et al.* 1994]. However the results are found identical up to a rescaling of time.

Simulations have been made for various systems sizes and an initial number of particles per site equals 10, on average. The results of the decay process are given in figure A.5. As one sees, all systems with size larger than 64×64 give the same decay exponent $\alpha = 0.875 \pm 0.005$. This value is in very good agreement with the value obtained by standard molecular dynamic simulations. Moreover, our algorithm is very efficient since the CPU time for the system of size 1024×1024 with initially 10^7 particles is about 58 seconds on a IBM-SP2 parallel computer with 10 processors. This is several order of magnitude faster than the molecular dynamics computation.

It is interesting to compare this results with those obtained with standard CA or LB simulation. A two-dimensional FHP[Chopard *et* Droz 1998] cellular

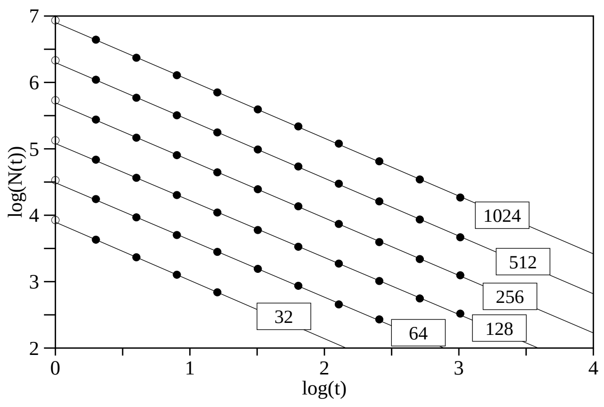


Figure A.5: Decay laws for the ballistic annihilation simulations, using the multiparticle lattice gas model. The various plots correspond to the lattice sizes indicated in the box. The decay exponent α is given by the slopes of the lines which are all within $x = 0.875 \pm 0.005$, except for the smallest lattice.

automata model with annihilaton gives an exponent $\alpha = 1/2$. This is the 1D exponent, which can be explained by the fact that, in this case, particle annihilation takes place before collisions can couple the two spatial dimensions.

For the standard, real-valued LB approach in 2D, annihilation is modeled by adding the terms $-F_i F_{i'}$ to the collision operation, where, again, i' denotes the direction opposite to i. Numerical simualtions then yield $\alpha = 1$, corresponding to the naive rate equation approach $\dot{\rho} \sim -\rho^2$.

Therefore, the multiparticle model clearly captures an subtle behavior of ballistic annihilation. It can be noticed in fig. A.6 that, despite the fast particle decay, multiparticle collisions are present all along the dynamics.

A.5 Conclusion

We have proposed a multiparticle algorithm that conciliates the advantage of both the CA and LB approaches: numerical stability, presence of fluctuations, little statistical noise, due to the large number of particles per site, and flexibility to tune model parameters. Although significantly slower than its LB counterpart, our dynamics can be implemented in an efficient way on parallel computers and is much simpler and more flexible than the other multiparticle models[Chatagny *et* Chopard 1993, Boghosian *et al.* 1997] proposed so far to extend the CA dynamics without an exclusion principle. In a slightly different spirit, we may also mention the recent multiparticle models by Masselot *et al* [Masselot *et* Chopard 1998a] and Malevanets *et al*[Malevanets *et* Kapral 1998].

Our model gives a remarkably good prediction for the ballistic annihilation problem and exhibits the expected hydrodynamical behaviors in the Poiseuille

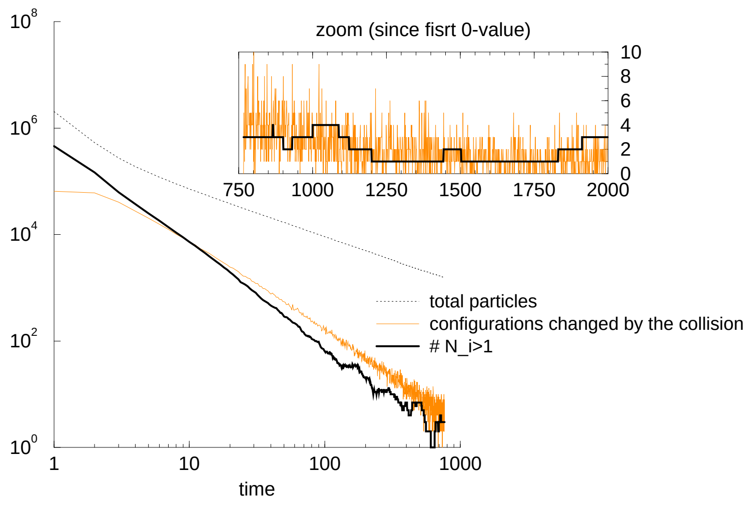


Figure A.6: Multiparticle collision frequencies during the ballistic annihilation simulation (256×256 cells). One can observe that even if there are a small number of particles in the whole system, rich collisions still occur and the multiparticle aspect ($N_i > 1$) remains. The scale is logarithmic for the early part of the simulation, and linear for the zoomed inset (as some experimental value may have become 0 for time step > 750).

and von Karman flows. A complete analytical derivation of the expression for the viscosity, taking into account the stochastic part of our algorithm is still needed.

Appendix B

Interactive experiments: a key to an efficient parameter space exploration

B.1 Why developing an interactive simulation tool

Exploring a vast multi-dimensional parameter space can be a formidable task (for the current work, the number of parameters has once reached 30, even if most of them have been thrown away since then). This task can even become toughest if some parameters are not direclty linked with physical quantities, or for which the experimentalist does not have *a priori* idea of the range where to look for something interesting to happen.

For example, once he has the idea of randomly erode solid particle with a probility ζ_p (cf. section 3.1.4), the experimentalist does not have the faintest idea of the values he should give to ζ_p ($O(1)$? $O(0.1)$? $O(.01)$? $O(0.001)$?). A too low ζ_p will involve fastidious computing times, wether a too high one will make all the particles move at the same time and therefore a steady state cannot be reached.

Parallel computers, offering large CPU resources thus short computing times, provides the basic tool for an extensive exploration. However, batch mode or off-line simulations return either *post-mortem* results or, more frustratingly perhaps, step-by-step states of a simulation that appears to be unperfect (but might be corrected tuning some parameters). Changing one parameter will anyway involve re-runing the simulation from scratch, thus waiting impatiently for the end of the overhead part of the simulation (for the fluid to get established ...), and will finally make the experimentalist have a new idea and re-run everything ...

B.2 Building simply an interactive simulation device

An on-line interactive simulation tool appears to be the solution. The purpose is to offer the user the opportunity to tune any parameter or performed some operations (extracting a fluid velocity profile, modifying the solid particles entry zone, saving the current results ...) during the simulation execution.

However, the simulation program (in Fortran 90, C, C++ ...) often already works, all the expected run-time modifications are only calls to procedures, and the programmer does not want to deeply modify his code to incorporate some heavy user-friendly interface. Moreover, defining such a user friendly interface would raise serious problems if the same code is to be compiled and ran on different machines. Last, but not least, the programmer shall spend his energy in developping his basic model rather than getting stucked in tedious imported library problems.

The solution we have adopted in the present work is the folowing: to send orders, the user simply type on-line some predefined commands (for example *'set_tau 0.504'* to set the relaxation time to 0.504, *'save_velocity_profile 3 20 4'* to save a vertical (dim=3) velocity profile at $x = 20$, $y = 4$ ...). At every iteration, the program verify if there is something on the standard input, and call the related subroutine if any. To save time, some frequent orders may be also called by a shortcut: *'set_tau 0.504'* can be replaced by *'t 0.504'*. A list of the available commands is presented in table B.1.

B.2.1 The code

The only problem is to check if there is anything to be read on the standard input. Fortran 90 does not offer this feature in its standard. The solution is therefore found with a small c program:

```
#include <stdio.h>
#include <sys/ioctl.h>

void nbbytestoget(l)
int *l;
{
ioctl(0, FIONREAD,l);
}
```

The subroutine *nbbytestoget(l)* can be called directly from fortran, l returns the number of bytes to be read on the standard input ($l = 0$ therefore means that nothing is to be read).

With some compilers, it can be requested to add the _ symbol after the C procedure to be called from fortran.

command	**arguments** (i:integer, r: real, l:logical, s string)	**action**
stop		stop the execution
set_istep	i	force the time step counter value to i (in batch mode, all the command planned during the intervall between the current time step and i are removed)
set_u_acc	l_1 l_2 l_3	$l_1 \Leftrightarrow$ fluid is accelarated from the $x = 1$ border,$l_2 \Leftrightarrow y = 1$ border ...
set_u_entry	r_1 r_2 r_3	set the entry velocity $\mathbf{u}_{entry}$ to (r_1, r_2, r_3)
set_tau	r	set the relaxation time τ to r
set_c_smago	r	set the smagorinski constant C_{smago} to r
set_gradient_top_0	l	$l \Leftrightarrow$ 0-gradient procedure is ran on the $z = 1$ and $z = nz$ layers (cf. section 2.3.1)
set_part_u_fall	r_1 r_2 r_3	set the solid particle falling speed $\mathbf{u}_{fall}$ to (r_1, r_2, r_3)
set_part_launch_n	i	launch i particles at each time step from the launching sites
set_part_jmp_thres	r	if the fluid velocity is r, the solid particle will be flown away with probability 1
set_part_solid_thres	i	set the thresehold number of frozen particles to solidify a site to i (cf. section 3.1.3)
set_part_erod_prob	r	set the erosion probability ϵ_p to r
set_part_cyclic	l_1 l_2 l_3	$l_1 \Leftrightarrow$ system is cyclic in the x-dimension, $l_2 \Leftrightarrow$ in the y-dimension ...
set_part_launch_area	i_{x1} i_{x2} i_{y1} i_{y2} i_{z1} i_{z2}	Set a new launching zone in the zone $(x, y, z) \in [i_{x1}, i_{x2}] \times [i_{y1}, i_{y2}] \times [i_{z1}, i_{z2}]$
solid_clean		erase the solid configuration
set_lnd_cylinder_rel	i r_1 r_2 r_3	if $i = 1$, builds a solid cylinder in the x-direction, centered at $(y = r_1 n_y, z = r_2 n_z)$
set_lnd_hex_rel	i_{x1} i_{x2} i_{y1} i_{y2} i_{z1} i_{z2}	solidify cells $(x, y, z) \in [i_{x1}, i_{x2}] \times [i_{y1}, i_{y2}] \times [i_{z1}, i_{z2}]$
set_lnd_2dprof_rel	i i_{x1} i_{z1} i_{x2} i_{z2} ... i_{xp} i_{zp}	define a solid profile, y-periodic, delimitated by the broken line of i points $(i_{x1}, i_{z1}) \rightarrow (i_{x2}, i_{z2}) \rightarrow \ldots (i_{xp}, i_{zp})$
save_u_velocity	s	save the velocity through out the domain as an *AVS/Express* field in file s
save_u_velocity_profile	s	save averaged fluid velocity versus z in the range $(x, y) \in [i_{x1}, i_{x2}] \times [i_{y1}, i_{y2}]$ in file s; the graph can later be ploted with *Xmgr* or *Gnuplot*
record_wind	i_n i_{x1} i_{y1} i_{z1} i_{x2} $i_{y2} i_{z2}$ ... i_{xp} i_{yp} i_{zp} i_t s	records in file s during i_t time steps the norm of the fluid velocity at positions (i_{x1}, i_{y1}, i_{z1}),(i_{x1}, i_{y1}, i_{z1}) ... (i_{xp}, i_{yp}, i_{zp})
save_u_vorticity	s	save the vorticity through out the domain as an *AVS/Express* field in file s
save_flying_part	s	save the flying particle density through out the domain as an AVS/Express field in file s
save_frozen_part	s	save the frozen particle depth as a multi-steps *AVS/Express* field in file s
save_landscape	s	Save the solid configuration (excluding particle deposit) as an *AVS/Express* UCD structure in file s

Table B.1: list of some commands to pilot the 3D application.

NB: if the simulation program is to be ran in parallel on remote machine (MPI, PVM), the standard input may be filtered by some daemon and not sent directly to the programmer task. It can therefore be necessary to implement a socket device, or a flag system for the front end process to broadcast the information.

B.3 Adding a user-friendly interface

An on-line command interface may appear obsolete to some user, even if it is comfortable to use it on a remote machine via a telnet connection. However, it is very simple to add a user-friendly interface without modifying one line of the above program.

A GUI interface may be designed using for example *TCL/TK*, as the one proposed to manage the 2D code running on the *Connexion Machine CM-200* in figure B.1. The interface starts the execution of the program and its output is piped onto the standard input of the slave program. Each button corresponds to a command and writes the appropriate line onto its output. The slave program does not even see that it obeys to this GUI rather than to directly the prompt.

In fact, the TCL/Tk interface we have developped can easily be adapted to any other program piloted by the same on-line command system. The list of command, their type (integer, character string ...) are stored in a pilot description file. Therefore, the adapting of the pilot interface to new commands is very simple. Moreover, the interface keep trace of all actions in a log file: therefore, the same expermients can be rerun, modifying only few parameters.

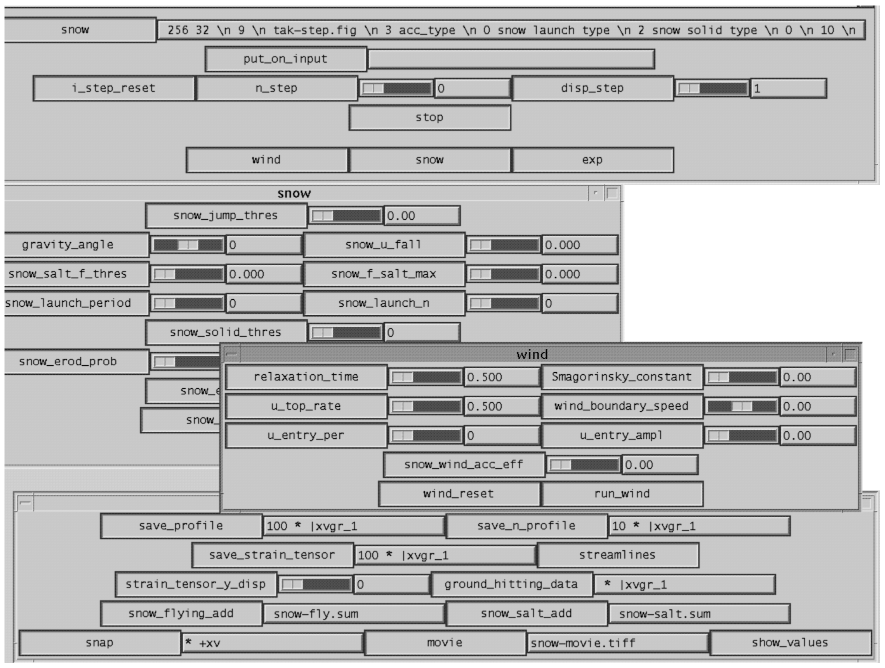
One inconvenient of the TCL/Tk interface is that it handles very slowly data flows (if the pilot asks the simulation program to return data for plotting a graphe, and if these data are to be sent to an external viewer -xmgr, gnuplot-, it should be avoided to make them transit by TCL/tk (as they would jam it): they should better be stored in a file by the simulation program, then a viewer should be launched opening this data file).

To build a more powerful interface, one could build the same device using a modern language, such as Java.

B.4 An all-purpose tool

Beside the main program of this current work, this interactive way of dealing with parameters have been used in many other applications where the investigation of the parameter space is important to find some interesting results: prey-predators, forest fire, sociological ... models.

For any of these codes, the only modifications was to link with the small precited C program, and to write a small procedure reading the input line command



```
snow                 256 32 \n 9 \n tak-step.fig \n 3 acc_type \n 0 snow launch type \n 2 snow solid type \n 0 \n 10 \n
n_step               0
disp_step            1
snow_jump_thres      0.00
gravity_angle        0
snow_u_fall          0.000
snow_salt_f_thres    0.000
snow_f_salt_max      0.000
snow_launch_period   0
snow_launch_n        0
snow_solid_thres     0
relaxation_time      0.500
Smagorinsky_constant 0.00
u_top_rate           0.500
wind_boundary_speed  0.00
u_entry_per          0
u_entry_ampl         0.00
snow_wind_acc_eff    0.00
save_profile         100 * |xvgr_1
save_n_profile       10 * |xvgr_1
save_strain_tensor   100 * |xvgr_1
strain_tensor_y_disp 0
ground_hitting_data  * |xvgr_1
snow_flying_add      snow-fly.sum
snow_salt_add        snow-salt.sum
snap                 * +xv
movie                snow-movie.tiff
```

Figure B.1: user friendly interface to pilot the 2D code running on the CM-200. This part is written in TCL/Tk, starts the main program and sends on its standard input the order corresponding to the button pushed. The list of all the features of this interface is stored in a file, thus offering an easy way to use the same TCL/Tk program for many different applications. However, the lack of compatibility between TCL/Tk versions can be a challenge in itself to produce a fully generic product. A pilot interface was only developped for the $2D$ code, as its purpose was to offer the programmer the opportunity of testing many parameters combinations interactively, and to explore a totally unknown space. In $3D$, as the world has already been partially explored by the $2D$ experiments, it was much more easy to rely only on batch procedures.

and executing the related action (for example, if it reads `set_tau` 0.5, it should execute `tau=0.5`).

B.5 Turning to an off-line script mode

As stated before, the off-line script mode does not offer the possibility of modifying interactively the execution. However, for production computations, it is important to have a way of launching larger scale experiments. These experiments will consists in several phases:

1. initializing the domain size, the base landscape,
2. setting the fluid parameters,

3. running the fluid model long enough to let the flow beeing established, as stated in section 2.3.1 (if high Reynold numbers are to be reached, it may be requested to run the experiment during some time steps with a higher viscosity before lowering it),

4. setting the solid particles parameters,

5. running the fluid and particle model,

6. at some time steps, saving the simulation state (deposit shape, fluid velocity ...).

All the command can therefore be stored in a file, preceded by the *at* command:

```
at 0 'set_plate 0. 1. 0. 1. 0. 0.' ! set a flat ground from x=1 to x = nx_tot,
                                   ! y=1 to y=ny, z=1
at 0 'set_tau 1.'
...
at 500 'set_erod_eff 0.01'         ! set ε_p to 0.01
...
at 10000 'save_deposit_profile'
...
at 20000 stop
```

The `at` *t* `'com'` command consist only in interpreting the command *com* at time step *t*. This command will be interpreted exactly as if it was typed in the interactive mode. The above file can simply be directed towards the standard input of the program (*i.e.* `a.out -batch < cmd.file`).

To store the command lines, the chosen technique was to make a linear dynamic list (in Fortran 90!) of the commands, containing the time step when to be executed and the command line. A variable stores the time step when the next command is to be executed; once this step is reached, the top command of the list is executed and removed from the list. Several commands may be executed during the same time step.

Another advantage of this off-line batch mode is to offer a very simple way of keeping trace of an experiment history and to replay it, or slightly modify it.

Appendix C

Visualizing the 3D model output: Avs/Express

Visualizing computations output can be a complex problem. In the case of solid particles transported by a flow, one may want to see the flow field (fluid velocity intensity, streamlines, streaklines, vorticity ...) and the solid particle distribution (either deposited or transported). Moreover, a time dependent representation of data appears to be important to capture the evolution of the system or to get a better feeling of its behavior.

For 2D simulations, displaying the information is simple as it can be spread among different windows on a computer screen. There isn't any dilemna about the choice of the point of view and the only problem can be to find a good colormap to clearly present the results. The coding of the display interface does not usually offer a great challenge, as it is only the mapping of 2D arrays onto a 2D screen (however, this may be a little more tricky with a parallel distributed execution, when all the processes, from remote machines, want to display their own data inside the same window in a coherent manner - see *http://cuiwww.unige.ch/~luthi/xlk/xlk.html*).

The situation is more complicated with 3D simulation domains, as computers can, for the time being, only display their data through a 2D screen or sheets of paper. Moreover, writing the code to present decently 3D data soon becomes a tedious work. For these reasons, we have chosen to use a commercial software, *AVS/Express* (Advanced Visual System *http://www.avs.com*). However, even if it is higly interactive, this software is tremendously complex as it has been designed to fullfill a very wide spectra of applications and adapting it to a given problem can request large amounts of time and patience.

We therefore present here a short introduction to AVS/Express[1] , and how to visualize through it the ouput of our 3D model. The aim of this chapter is only

[1]the version we have used is *AVS/Express Visualization Edition, Version: 4.2* running on a *Sun Ultra 1* under Solaris 5.6, but the features presented here are more or less generic.

to make a further user of this software (for the same kind of problems) save time and energy. As very little (if any) literature exists, despite lecture notes from some expensive courses, this appendix should not be totally unuseful to some academic users.

We will present the use of basic data structures (uniform (multistep) fields and UCD), how to write them from the simulation program and tools to represent them (slicers, streamlines, isosurfaces ...) with the *AVS/Express* software, in the way they have shown to be useful for our application. The topics of this section are more extensively developed in the user guide *Visualizing your data with AVS/Express.*

C.1 A simple example

AVS/Express reads different file formats and allows the user to display the data in a *viewer* through a *network* of *modules.* For example, the chain `Read_Field`→`Orthogonal_Slicer`→`Is` reads a field contained in a file, takes a slice, computes the isolines and displays the results on a graphic window. Every module has its own parameters (a file name for `Read_Field`, a plane dimension and an offset for `Orthogonal_Slicer` ...) and this example is contained in an *application* (format **.v*, which is a text file interpreted by AVS/Express, but can easily be understood in its main lines by the programmer) as shown in figure C.1.

The data is saved in a file by the simulation program and later read by *AVS/Express.* To be more efficient, files can be saved during the simulation and read at regular time intervals by the application. This method is even more easier if the file system is shared by the simulation computer system and the workstation running *AVS/Express*[2]

C.2 The *field* format

The *field* files are the main channel between an application and *AVS/Express*[3]. As every feature of this software, this data structure can become complex but, once again, we will restrict us to demonstrate how it can be used to store data from our application, *i.e.* from a uniform 3D domain.

A field file is split in two entities:

- the header,
- the data.

[2]**NB:** the byte representation of real and integer are inversed in Linux compared to Unix. Therefore, if the simulation is ran on a Linux platform and *AVS/Express* on a Unix one, the data file must be treated.

[3]For further information about the field format, read the chapter *Field* data type in the user guide *Data visualization kit*

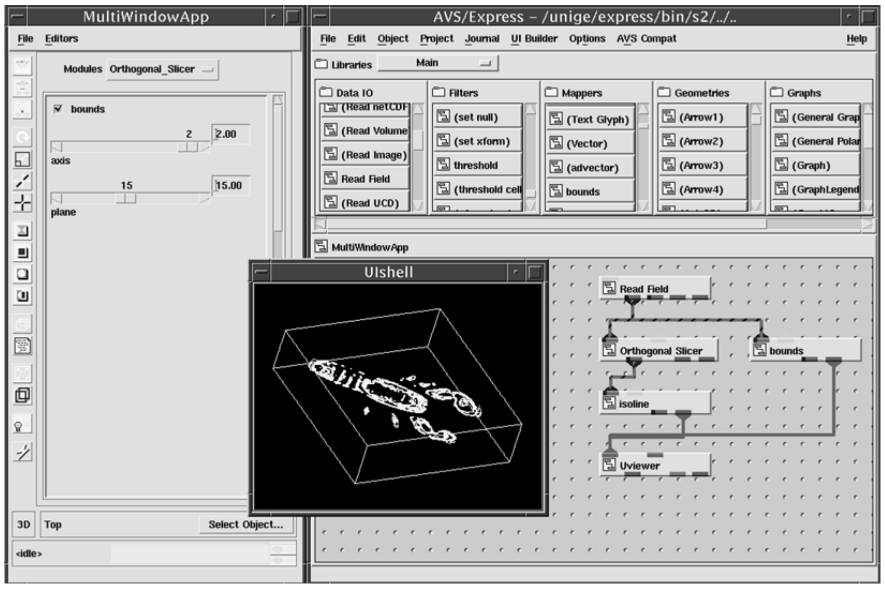


Figure C.1: A snapshot of a simple application in the *AVS/Express* environment. The network editor in the righ hand side, containing libraries of modules and the graph of modules linked together; the left hand side contains informations about the enlighted module; the black graphic window corresponds to the `UViewer` module of the network (the point of view can be modified using the tools - zoom, translation, rotation - displayed on the extreme left of the figure).

C.2.1 A practical example

To illustrate this format, we can take a practical case. Let's consider a 3D domain $n_x \times n_y \times n_z$ (where $n_x = 100$, $n_y = 30$ and $n_z = 50$). Throughout this domain, a variable containing the 3 components of the fluid velocity is defined. In Fortran 90, it could be defined as:

```
real,dimension(3,50,30,100)::vel
```

where the dimension are inversed in prevision of the parallel distribution of the workload, where the domain is split among the $x-$dimension (which is often the largest) as explained in section 4.3.2.

The *field* data file (**.fld.dat*)

The data file contains the values of the stored field, its storing order is not unique and has to be explicited in the header file. However, when reading an array, the first index varies most quickly (fortran style).

In the current example, the variable `vel` is saved in an unformatted file (writing on disk directly the byte representation of the data, to save space) in the following manner:

```
open(10, file='ex-1.fld.dat',form='unformatted')
do l=1,3
  write(10)(((vel(l,k,j,i),i=1,100),j=1,30),k=1,50)
enddo
```

It could have been saved in some other ways, but this one will store the values of `vel` in the following order:

```
@@@@ vel(1,1,1,1) vel(1,1,1,2) vel(1,1,1,3)
... vel(1,1,1,100) vel(1,1,2,1) ... vel(1,50,30,100)@@@@
@@@@ vel(2,1,1,1) vel(2,1,1,2) vel(2,1,1,3)
... vel(2,1,1,100) vel(2,1,2,1) ... vel(2,50,30,100)@@@@
@@@@ vel(3,1,1,1) vel(3,1,1,2) vel(3,1,1,3)
... vel(3,1,1,100) vel(3,1,2,1) ... vel(3,50,30,100)@@@@
```

where there is neither space or return characters in this file, as the values are stored unformatted by their byte representation (4 bytes as `vel` is declared as `real`).

The loop inside the above `write` statement is performed to inverse the indice order (we could have saved `vel` directly through `write(10) vel(l,:,:,:)` if `vel` had been defined in the classic way as `real,dimension(3,100,30,50)::vel`). Moreover, the chains `@@@@` are four bytes blocks which appear before and after any written data in the Fortran unformatted mode (there is no such features in C). We will have to take into account this strings when reading the data from *AVS/Express*.

The ***field header*** file (**.fld*)

The header contains information about the field, *i.e.* the number of dimensions, the kind of data and its location ... It is a short ASCII file, such as, for the current example, shown in table C.1

Multi-steps fields

As getting only a snapshot of a simulation is often not enough to catch the behavior of a system, the experimentalist often needs to see its evolution. A solution is therefore to save regularly the system state in a field. Instead of tediously opening by hand a list of field files, *AVS/Exprees* offers the possibility of building *multi-steps fields*.

Let's modify the previous example, saving the system every 1000 steps in 500 files labeled `ex-1.001.fld.dat`, `ex-1.002.fld.dat` ... `ex-1.500.fld.dat`. We therefore need to write a more complete header file in the following manner:

```
# AVS field
# wind velocity every 1000 steps
ndim = 3
```

`# AVS field`	
`# wind velocity`	Comments lines must start with `#`
`# max:  0.1943228 min:  -0.1513349`	
`ndim= 3`	Number of computational dimensions
`dim1= 100`	n_x
`dim2= 30`	n_y
`dim3= 50`	n_z
`nspace= 3`	Number of physical dimensions
`veclen= 3`	Number of data stored at each nodes (3 velocities components)
`data=float`	Type of the data
`field=uniform`	`uniform`, as the domain uniformly maps the domain
`variable 1 file=ex-1.fld.dat filetype=binary skip=4 stride=1`	The first variable (the array containing the first component over 3, as `veclen` is stated), is stored in file `ex-1.fld.dat`; the 4 first bytes of the file should be skiped (`@@@@` fortran tag); the data are adjacent,*i.e.* all the data should be read continuously (if the first component was stored on every 3 reals - `write(10)vel` -, we would have `stride=3`)
`variable 2 file=ex-1.fld.dat filetype=binary skip= 600012 stride=1`	`600012` bytes must be skipped before reading the second components ($100 \times 30 \times 50 \times 4$(=size(real))$+3 \times 4$(=three preceding `@@@@` tags))
`variable 3 file=ex-1.fld.dat filetype=binary skip=1200020 stride=1`	

Table C.1: *field header* file for the mono-step example described in paragraph C.2.1

```
dim1 =  100
dim2 =    30
dim3 =    50
nspace = 3
veclen = 1
data = float
field =uniform
nstep =   500
DO
time value =      step_1000
variable 1 file=ex-1.001.fld.dat filetype=binary skip=4 stride=1
variable 2 file=ex-1.001.fld.dat filetype=binary skip=   72012 stride=1
variable 3 file=ex-1.001.fld.dat filetype=binary skip=  144020 stride=1
EOT
time value =      step_2000
variable 1 file=ex-1.002.fld.dat filetype=binary skip=4 stride=1
variable 2 file=ex-1.002.fld.dat filetype=binary skip=   72012 stride=1
variable 3 file=ex-1.002.fld.dat filetype=binary skip=  144020 stride=1
EOT
.
.
.
repeated (incrementally) 500 times
.
.
.
time value =      step_500000
variable 1 file=ex-1.500.fld.dat filetype=binary skip=4 stride=1
variable 2 file=ex-1.500.fld.dat filetype=binary skip=   72012 stride=1
variable 3 file=ex-1.500.fld.dat filetype=binary skip=  144020 stride=1
EOT
```

The *AVS/Express read field* menu is therefore slightly modified when opening such a file, as options concerning the time steps are proposed (running once the whole simulation, displaying directly the step x ...). Note that, for conveniency, all the data files can be appended in one big file (the skip options should consequently be adapted).

C.3 The Unstructured Cell Data format (**.inp*)

Beside the computed data, there are often surrounding a static configuration. For example, in the case of the flow around a fence, one would like to see the fluid flow, but also the solid structure.

It is possible to render such a solid configuration via a hierarchical field file, as this solid has once been defined as a set of simple geometrical elements. However, the *UCD* (*Unstructured cell data* file format proposes a much simpler way for such definitions.

As we did with the *field* file format, we won't propose here an extensive overview of te *UCD* format, but rather an introduction with enough informations to reproduce our results.

C.3.1 Description

A *UCD* file contains:

1. a list of *nodes* (3D space points);
2. a list of *cells* (build with *nodes*);
3. informations associated with each node (temperature, pressure...);
4. informations associated with each cell.

C.3.2 Example

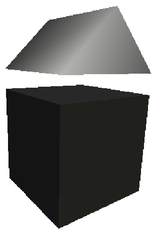


Figure C.2: This picture can be considered as 6 quadrilaterals to build the cube, plus 2 more quadrilaterals and 2 triangles to build the prism (this second part could also have been considered as a whole cell). The file describing this object is given in table C.2.

Further examples can be found in the *AVS/Express* directory */avsdir/data/UCD* (where even multistep *UCD* can be built).

C.4 Building an application from modules

In section C.1, we have briefly shown how *AVS/Express* works, assembling a chain, or more generally an oriented non-cyclic graph of *modules.* This network is stored into an *application* (**.v* file), interpreted by the *AVS/Express* software. This application can be hacked via a text editor or more simply interactively built via the network editor, already presented in figure C.1.

Therefore, a user may build an application dedicated to his specific needs. We will present here several short examples, focusing on problems we have faced visualizing our results.

```
14 10 0 2          the object contains 14 nodes, 10 cells
                   no information per nodes, 2 per cells
1 0 0 0            nodes are labelled, spatial coordinates are given
2 0 0 1
3 0 1 0
4 0 1 1
5 1 0 0
6 1 0 1
7 1 1 0
8 1 1 1
9 0 0 1.2
10 0 1 1.2
11 1 0 1.2
12 1 1 1.2
13 0 .5 1.7
14 1 .5 1.7
1 0 quad 1 2 4 3   cells are labelled and described
2 0 quad 1 2 6 5   cell #2 is a quadrilateral,
                   built from nodes 1, 2, 6 and 5
3 0 quad 1 3 7 5
4 0 quad 8 7 5 6
5 0 quad 8 7 3 4
6 0 quad 8 6 2 4
7 0 tri 9 10 13    cell #7 is a triangle,
                   built from nodes 9, 10 and 13
8 0 tri 11 12 14
9 0 quad 9 11 14 13
10 0 quad 10 12 14 13
2 1 1              for each cell, there are two informations (in this example)
                   either one or the other can be displayed via a colormap
temperature, t     first a temperature
porosity, phi      second a porosity
1 .1 .7            for cell #1, temp=.1, porosity=.7
2 .2 .8
3 .3 .4
4 .4 .2
5 .5 .6
6 .6 .2
7 1. .3
8 1. .3
9 1. .3
10 1. .3
```

Table C.2: a simple *UCD* file, to build the object displayed in figure C.2 (italic comments should be removed in the effective file). The informations attached to the cells (or to the nodes) can be individually extracted through the module `Extract_Cell_Component` (see section C.4.2. The second number, in the cell description lines, can be used to group the cells into *cell sets.*

C.4.1 Modules

4.2.2 Modules are pieces of codes with external *parameters*, *input* and *output ports.* For example, the `Orthogonal_Slicer` module displayed in the example of figure C.1 takes for input a *3D* field, for parameters the orientation and the position of the slice, and outputs a new field containing the extracted slice.

Modules are *linked* to each other: the output port of a module can therefore be related to the input port of another one, assuming that their types are coherent (an integer can be sent to a port querying a real scalar, but a field output cannot be sent directly to a viewer). When building an application with the network editor, the color of a port indicates its type.

There are several ways of modifying the parameters of a module:

1. beside the network editor, an application window allows us to see and to modify them (with a user friendly interface).

2. Inside the network editor, the icon of a module can be extended by double-clicking on it; each parameter can therefore be edited by hand.

3. In a more powerful manner: using the network editor, instead of editing it, a parameter can be *exported* to an input/output port. This port can therefore be linked to an other one[4]. For example, when visualizing two multi-steps fields (fluid velocity and solid particle deposit), the `current_step` of the first field can be attached to an output port, exported and linked to the second field `current_step`, itself attached to an input port; therefore, modifying the `current_step` of the first parameter will automatically synchronize the second one.[5]

4. Further on, the user may notice that linking two parameters replaces the destination one by an expression, more or less tedious, pointing to the source. This expression can be edited and arithmetically modified (two `Orthogonal_Slicer`'s on the same direction but at a constant distance ...).

C.4.2 Several useful modules

Once again, this appendix does not want to figure as a complete guide of *AVS/Express*, and we will present here a panel of the most important modules we have met.

Modules are stored within libraries such as `Data_IO`, `Filters`, `Mappers`, `Graphics` ... A module can be searched for among these libraries using the menu *Objects/Search in all libraries.*

`UViewer, UViewer2D, UViewer3D`: related to a graphic window, this module is usually the end of a network.

[4]Before linking, the destination parameter should be erased, showing a `?` in place of its value.

[5]In version 4.2, a bug does not allow us to link these two `current_step` field directly; the user must put a `copy_on_change` module between both.

`Read_Field`: reads a file containing a field (the file name can be browsed). The output can either be a field or directly sent to a viewer. For multi–step fields, parameters such as `current_step`, `total_steps` are available.

`Combine_vect, Extract_Component, Magnitude`: when, as in the example of section C.1, a field contains a vector information about each grid point, it can be necessary either to extract one of those components or to combine them and to extract the magnitude of this local information.

`Read_UCD`: reads a file containing a UCD structure (described in section C.3) and output either a field or a scene to be connected to a viewer. An image can be mapped onto the so generated mesh via the module `Read_Image` and `Texture_Mesh`.

`Isosurface, Isovolume, Isolines`: as indicated by their names, these modules take a field as input and output a new one containing only an isosurface, isovolume or isolines. Their parameter are therefore the level the iso-value for the two first ones, and the number and range of the iso-values for the last one.

`Image_Capture`: linked to a viewer, records every display. This module is used to build *mpeg* movie files from the views. Moreover, the module `Animator` allows to interpolate a film between given views (for example, giving a first orthoslice at $x = 1$ and a second one at $x = 100$, it can build 30 transition images for the intermediary slices) and later build *mpeg* file.

Many other modules exist, and a good gallery to try them is to explore the `Examples` library distributed with the software.

C.4.3 Two examples

The modules presented in the previous subsection are exploited in two short examples displayed in figure C.3 and C.4. These networks can of course be enlarged or merged to build a complex application.

Conclusion

AVS/Express is definitely a very powerful tool for 3D visualization. The price to pay is a good workstation with memory, and some patience to get inside its “way of thinking”. Bugs or disfunctions are not rare but have always been solved in our case with the distributor’s hotline.

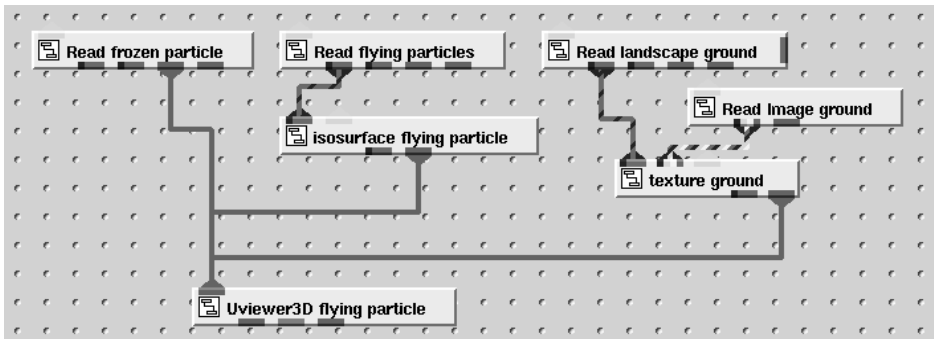


Figure C.3: Frozen and flying particles are stored in two fields. The first one is directly sent to the viewer while an isosurface of the second is extracted. A module `Read_UCD` is employed to read a UCD representation of the basic solid configuration, and a texture is mapped onto it. The position of the modules is not important, and should be arranged in order to facilitate the reading. Every modules parameters can be interactively tuned via a dedicated window (left hand side on figure C.1), or more explicitely programmed by the user (see section 4.2.2).

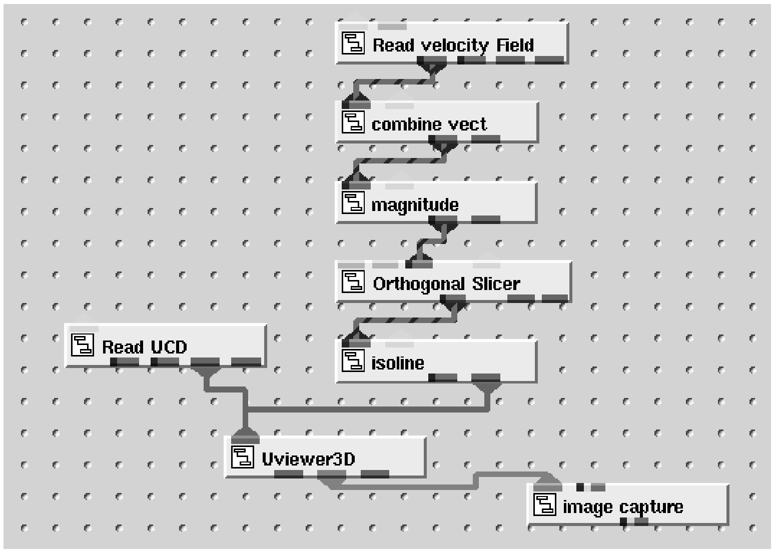


Figure C.4: Solid configuration is represented by a UCD field, but this application is dedicated to display an orthoslice of isolines of the norm of a field such as that decribed in section C.1.

Appendix D

A MPI benchmark

The Message Passing Interface (MPI) presented in section 4.3.2 is available on the majority of platforms as a high level library. Therefore, it is natural to make some comparisons between those hardware/software packages, to get a more precise opinion about their parallel performances.

We will present in this appendix several small MPI applications, each of them focusing on a special communication aspect, termed: a) ping-pong; b) shifting data; c) meli-melo; d) burning token; e) TLM code (simple lattice model for wave propagation).

These benchmarks have been ran on four different machines, with the following characteristics:

	Entreprise 10000 Sun	**Pentium II Cluster**	**Pentium III Cluster**	**Sun Ultra 5 Cluster**
Processor	Ultra	Pentium II	Pentium III	Ultra 5
Frequency	400Mhz	266Mhz	500Mhz	270Mhz
# nodes	12	32	3??128M	128M
Data cache	8M	1M	?M	512K
network	**Shared mem.**(crossbar)	Fast Ethernet (Extrem 48)	Fast Ethernet (Extrem 48)	Scali Double ring
Price	expensive	affordable	affordable	medium
Compiler	Nag f95 + Sun mpi	NAG f95 + mpich	NAG f95 + mpich	Apogee f90 + Scali MPI libraries

For each of those machines, the fastest compiler available was used, *i.e.* NAG Fortran 95 version 4.0 (except for the Ultra 5 cluster, for some incompatiblity with the Scali MPI implementation).

D.1 Ping-pong

Packets of growing sizes are bouncing between two tasks, and the aim of this test is to measure the maximum bandwidth minimum latency for each architecture.

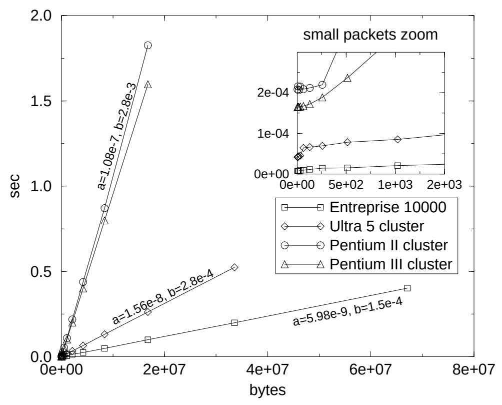


Figure D.1: **ping-pong**. Time displayed for a one way trip. The time $t(n)$ to transfer n bytes obeys well the law $t(n) = an + b$, where b is the latency and $1/a$ the bandwidth. However, the behavior for short buffers is not so linear (two regimes). On the other extrem, the Sun Entreprise 10000 can transfer decently up to 500 Mbytes buffers, when the other machines have dramatically long times for the values of n not plotted in this graphe. The inset figure shows a zoom of the global graphe at the origin, *i.e.* the communication times for small packets.

Figure D.1 exhibits a performance order: Entreprise 10000, Ultra cluster and Pentium cluster. It also shows how, with the same ranking, machine are able to manage in a efficient way larger and larger messages.

D.2 Shifting data

Contrary to the precedent experiment where only two nodes were involved, packets of growing sizes are sent between neighboring processes. The aim of this simulation (figure D.2) is to get the latency and bandwidth when all the tasks are communicating in a regular manner (conflict should easily be managed).

The same ranking as for the precedent experiment is respected. However, the agp between the machine performances decreases, as Entreprise's bandwidth is worsen by a factor 6, Ultra 2 cluster, by 4, and the Pentium's only by a factor 2. Latencies are also damaged.

D.3 Meli-melo

In this experiment, every node sends 100 messages (1 integer) to random destinations, and every node receives all its intended messages. The topic is to saturate the network with a massive load of small and unstructured (no route can be privilieged) messages

The sending and receiving times are measured separately and a barrier is set between the two parts to avoid spurious effects. Each experiment is repeated a hundred

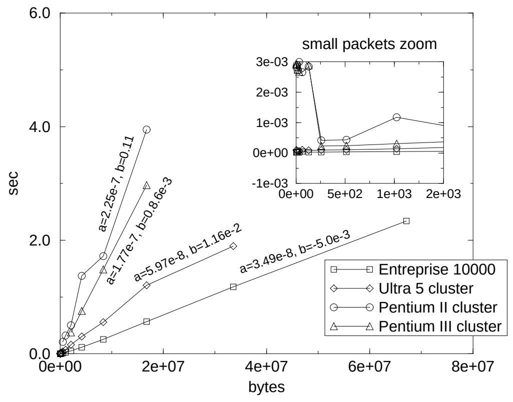


Figure D.2: **shifting data**. Packets of growing sizes are shifted. The time $t(n)$ to transfer n bytes still obeys well a law $t(n) = an + b$. a and b are different as the network is totally loaded (all processors are participating). Comparing with ping-pong performances, Entreprise's bandwidth is worsen by a factor 6, Ultra 2 cluster, by 4, and the Pentium's by 2. Casualities on the latency are even worse.

times and the results are averaged in figure D.3.

This expermient is very demanding for the network, and a switch (such as with the Pentium cluster) acts as a serious bottleneck and produces very noisy results. On the opposite, the double ring interconnecting network of the Ultra cluster and the crossbar switch of the Entreprise 10000 handle this situation much more efficiently. The hierarchy between the four machines is still respected.

D.4 Burning token

A token (integer) is initialised to 1000 by task 0. Every task who get the token decrements it by 1 and sends it to a random task. The game ends by a global communication when the token reaches 0.

The goal is to build irregular communications (as the token as a random path) without saturating the bandwidth (figure D.4).

D.5 TLM

A classic Boltzmann model to model radio wave propagation [Luthi 1998]. Its simplicity (few floating operations and synchronous communications to neighbours on a 2D matrix) have made this model the simplest one for real life applications, and a recognized benchmarking application as it focused on Mflop/s and network bandwidth performances.

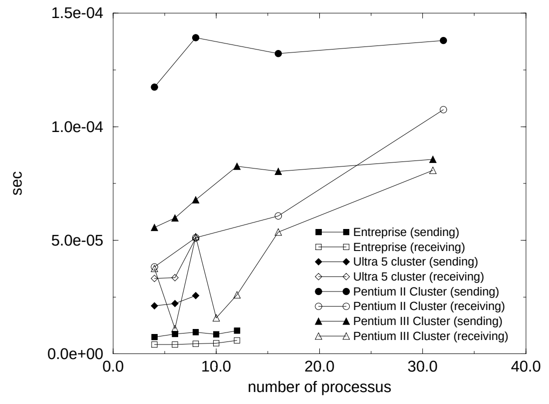


Figure D.3: **meli-melo**. Time is displayed for the averaged (one byte) message emission/reception. The previous hierarchy among the machine is respected. The Pentium clusters produces very noisy results, as a switch acts as a serious bottleneck on this irregular problem. As message passing are only *memove* on the shared memory Entreprise, and thanks to the fast switch between CPU and memory, it is natural to get there very high performances.

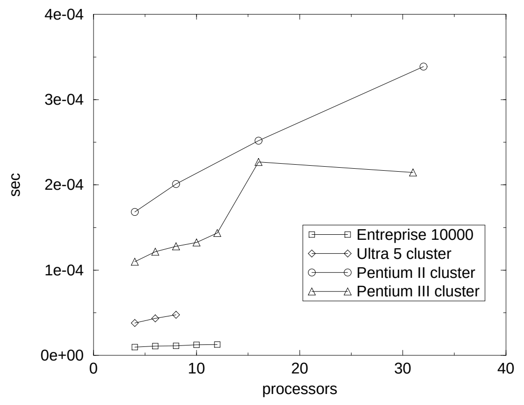


Figure D.4: **burning token**. Time is displayed for the averaged (one byte) message emission/reception. The previous hierarchy among the machine is respected. Even if the is one message travelling at once, the communication is slower has they are done sequentially (in the previous experiment, all the messages were sent at the same stage)

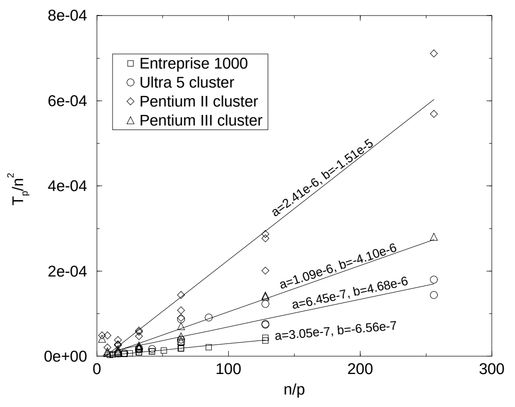


Figure D.5: **TLM**. n is the size of the side of the square domain on which the model is computed. There are n^2 cells, computed for n time steps on p processors. For each processor, each time steps is n^2/p cell computations + n data comunication (only boundary layers are sent to the neighboring tasks). We can write the simulation time as $T_p/n = an^2/p + bn \Rightarrow T_p/n^2 = an/p + b$, where a measure the flop/s and b the network bandwidth. It is interessant to note that the Pentium III cluster exhibits two strongly different regimes, for small and larger domain per node (it sticks to the ultra 5 cluster for $n/p < 64$), certainly because of some high performance cache management.

The experiment is ran for different times and numbers of processors and results are displayed in figure D.5. From this experiment, it is possible to fit the performances and to extract the Flop/s and the network bandwith for each machine and therefore to compare them according these two crucial parameters.

Even if, for the same number of processors, it can be seen how the hierarchy is observed. However, it can also be observe how a Pentium III cluster with three times more CPU can be faster than an Entreprise on this problem (and more generally, on many scalable problems).

D.6 Conclusions

Benchmarking machines can soon become a very complex task, as many factor are involved in the process [Hockney 1996]. However, in this appendix, our aim was to focus on different MPI performances on "working configurations", *i.e.* with classical programming and neither fancy nor non-portable hacking tricks.

Figure D.6 shows how different compilers (or even programming method, such as indexed Fortran 77 or array Fortran 90 notations) can influence the results.

Meanwhile, if the best compiler is chosen, a clear hierarchy emerge from these benchmarks: 1) Entreprise 1000, 2) Ultra 5 Cluster, 3) Pentium III cluster and 4) Pentium II cluster. This ranking is not surprising: it is coherent with theoritical

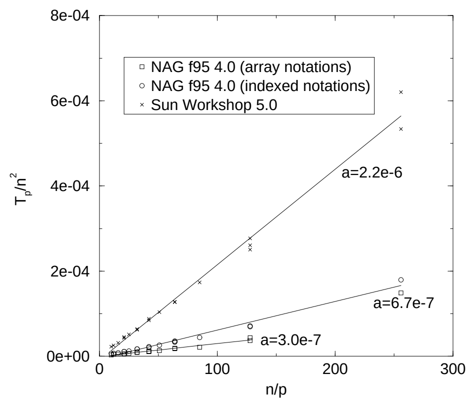


Figure D.6: the TLM benchmark on the Sun Entreprise 10000, with different F90 compilers and programming styles. The Sunpro compiler should definitely be avoided. It is interesting to note that we gain a factor 2, with NAG, by writing array operation $a = b + c$ and not $i = 1..n, j = 1..n, a(i,j) = b(i,j) + c(i,j)$

performances of the hardware (CPU's and interconnecting networks), but also with the price of the machine.

A much more interesting observation is how a cluster of Pentium with a fast-ethernet switch (*i.e.* a rather cheap parallel computer) can beat a very expensive one (for example the Entreprise 10000, but results were the same with an IBM SP2) by getting more CPU.

The main requirement in this case is to have an efficiently parallelized code, no to be limitated by the poor network. Another major advantage of Pentium cluster in a world where high performance computing resources are always overloaded, is that its price allows smaller research groups to acquire one for their own purpose and to have it dedicated to their own needs.

Appendix E

Publications

A list of the main publications, in the domain of snow transport by wind, but also in other topics:

Chopard B., Luthi P. and Masselot A., submitted, 1998. Cellular automata and lattice boltzmann techniques: an approach to model and simulate complex systems. *Advances in Physics.*

Chopard Bastien, Masselot Alexandre and Droz Michel, 1998. A multiparticle lattice gas model for a fluid. Application to ballistic annihilation. *Phys. Rev. Lett.*, **81**, 1845–1848.

Chopard Bastien, Masselot Alexandre and Droz Michel, 1998. Kinetic of the two-dimensional ballistic annihilation: a multiparticle lattice gas study. *Computer Physics Communications.*

Galam S., Chopard B., Masselot A. and Droz M., 1998. Competing species dynamics: Qualitative advantage versus geography. *Eur. Phys. J. B*, **4**, 529–531.

Masselot A. and Chopard B., 1998. A lattice Boltzmann model for particle transport and deposition. *Europhys. Lett.*, **42**, 259–264.

Masselot A. and Chopard B., 1998. A lattice boltzmann model for snow transport and deposition. *Europhysics Letters*, **42**(3).

A. Masselot and B. Chopard,1996 Cellular automata modeling of snow transport by wind. *In J. Dongarra, K. Madsen, and J. Wasniewski, editors, Lecture Notes in Computer Science*, Vol. 1041, pages 429-435. Springer, 1996.

Masselot A., March 1995. *A massively parallel approach to model snow transport by wind.* Technical report, University of Geneva.

Participations to several international conferences, where the publications were only proceedings: Para95 (Copenhagen), Fluid and Particle Interactions (Davos, 1996), VecPar98 (Porto), LGA'98 (Oxford), Fluid and Particles Interactions (Santa Fe, 1999).

Bibliography

[Anderson *et* Bunnas 1993] Anderson R.S. and Bunnas K.L., 1993. The mechanics of aeolian ripple sorting and stratigraphy as visualized through a cellular automaton model. *Nature*, **365**, 740–743.

[Anderson *et* Haff 1988] Anderson R.S. and Haff P.K., 1988. simulation of eolian saltation. *Science*, **244**, 820–823.

[Anderson *et* Hallet 1986] Anderson R.S. and Hallet H., 1986. Sediment transport by wind: toward a general model. *Geological Society of America Bulletin*, **97**, 523–535.

[Anderson *et* McDonald 1990] Anderson R.S. and McDonald R.R., 1990. Bifurcations and terminations in eolian ripples. *Eos*, **71**, 1344.

[Anderson 1986a] Anderson R.S., 1986. Eolian sediment transport as a stochastic process: the effects of a fluctuacting wind on particle trajectories. *Journal of geology*, **95**, 497–512.

[Anderson 1986b] Anderson R.S., 1986. Erosion profiles due to particles entrained by wind: Application of an eolian sediment transport model. *Geological Society of America Bulletin*, **97**, 1270–1278.

[Anderson 1986c] Anderson R.S., 1986. A theoretical model for eolian impact ripples. *Geological Society of America Abstractswith Programs*, 527.

[Anderson 1989] Anderson R.S., 1989. saltation and suspension of snow by wind: a general theory. abstract. *Annals of glaciology*, **13**, 294.

[Anderson *et al.* 1991] Anderson R.S., Sørensen M. and willetts B.B, 1991. A review of recent progress in our understanding of aeolian sediment transport. *Acta Mechanica*, **[suppl] 1**, 1–19.

[Anno 1984] Anno Y., 1984. Requirements for modeling of a snowdrift. *Cold Regions Science Technology*, **8**, 241–252.

[Bang *et al.* 1994] Bang B., Nielsen A., Sundsbø P.A. and Wiik T., 1994. Computer simulation of wind speed, wind pressure and snow accumulation around buildings (snow-sim). *Energy and Buildings*, **21**, 235–243.

[Bathnagar *et al.* 1954] Bathnagar P., Gross E.P. and Krook M.K., 1954. *Physical Review Letters*, **94**(511).

[BN *et al.* 1993] Ben-Naim E., Redner S. and Leyvraz F., 1993. *Phys. Rev. Lett.*, **70**, 1890.

[Bogard *et* Tiederman 1987] Bogard D.G. and Tiederman W.G., 1987. Characteristics of ejections in turbulent channel flow. *Journal. Fluid. Mech.*, **179**, 1–19.

[Boghosian *et al.* 1997] Boghosian B.M., Yepez J., Alexander F.J. and Margolus N.H., 1997. Integer lattice gases. *Phys. Rev. E*, **55**, 4137–4147.

[Castelle 1995] Castelle T., 1995. *Transport de la neige par le vent en montagne: approche expérimentale du site du col du Lac Blanc.* PhD Thesis, EPF Lausanne, Switzerland.

[Castelle *et al.* 1992] Castelle T., Hertig J.A. and Fallot J.M., 1992. *Protection des routes alpines contre les congères.* Technical report, LASEN, Swiss Federal Institute of Technology.

[Chatagny *et* Chopard 1993] Chatagny Rodolphe and Chopard Bastien, 1993. Multiparticle lattice gas models for hydrodynamics. *Science on the Connection Machine System: Proceedings of the Second European CM Users Meeting*, éd. par Alimi J.-M., Serna A. and Scholl H. Thinking Machines Corporation, 239–47.

[Chen 1999] Chen Yu and al. (édité par), 1999. *The 9th International Conference on the Discrete Simulation of Fluid Dynamics.* Tokyo, Computer Physics Communications. To appear.

[Chen *et al.* 1998] Chen Hudong, Teixera Chris and Molving Kim, 1998. Realization of fluid boundary conditions via discrete boltzmann dynamics. *International Journal of Modern Physics*, **9**(8), 1281–1292.

[Chopard *et* Droz 1998] Chopard B. and Droz M., 1998. *Cellular automata modeling of physical systems.* Cambridge University Press.

[Chopard *et* Luthi 1994] Chopard Bastien and Luthi Pascal, 1994. Reaction-diffusion cellular automata model for the formation of liesegang patterns. *Phys. Rev. Lett.*, **56**, 1505.

[Chopard *et al.* 1994]	Chopard B., Frachebourg L. and Droz M., 1994. Multiparticle lattice gas automata for reaction-diffusion systems. *Int. J. of Mod. Phys. C*, **5**, 47–63.
[Chopard *et al.* 1998a]	Chopard B., Luthi P. and Masselot A., submitted, 1998. Cellular automata and lattice boltzmann techniques: an approach to model and simulate complex systems. *Advances in Physics.*
[Chopard *et al.* 1998b]	Chopard Bastien, Masselot Alexandre and Droz Michel, 1998. A multiparticle lattice gas model for a fluid. application to ballistic annihilation. *Phys. Rev. Lett.*, **81**, 1845–1848.
[Clappier 1991]	Clappier A., 1991. *Influence des particules en saltation sur la couche limite turbulente.* Technical report, LASEN, Ecole polytechnique Fédérale de Lausanne (Switzerland).
[Cornell *et al.* 1991]	Cornell S., Droz M. and B.Chopard, 1991. Role of fluctuations for inhomogeneous reaction-diffusion phenomena. *Phys. Rev. A*, **44**, 4826–4832.
[Cornish 1914]	Cornish V., 1914. *Waves of sand and snow.* London, T. Fisher Unwin.
[Counihan *et al.* 1974]	Counihan J., Hunt J.C.R. and Jackson P.S., 47 1974. wakes behind two-dimensional surfaces obbstacles. *Journal of fluid mechanics*, 657–664.
[Cousteix 1989]	Cousteix J., 1989. *Turbulence et couche limite.* Toulouse, Cepadues éditions. Chap. 5.4.1 pp196-199, chap 6.1.1 229-233.
[Delannoy 1997]	Delannoy C., 1997. *Programmer en Fortran 90.* Eyrolles.
[Doolen 1990]	Doolen G. (édité par), 1990. *Lattice Gas Method for Partial Differential Equations.* Addison-Wesley.
[Dupuis *et* Chopard 1999]	Dupuis Alexandre and Chopard Bastien, 1999. An object oriented approach to lattice gas modeling. *Future Generation Computer Systems.* Best selected papers from HPCN'99, to appear.
[Dupuis *et* Chopard 2000a]	Dupuis Alexandre and Chopard Bastien, 2000. Lattice gas modeling of scour formation under submarine pipeline. *Journal of Hydraulic Engineering.* submitted.

[Dupuis *et* Chopard 2000b]	Dupuis Alexandre and Chopard Bastien, 2000. Lattice gas simulation of sediment flow under submarine pipelines with spoilers. *Hydroinformatics 2000.* To be presented.
[Dyunnin 1959]	Dyunnin A.K., 1959. Fundamentals of the theory of snowdrifting. *Isvest. Sibirsk. Otdel Akad. Nauk SSSR*, **12**, 11–24. National Research Council of Canada Technical Translation 952.
[Elskens *et* Frisch 1985]	Elskens Y. and Frisch H.L., 1985. *Phys. Rev. A*, **31**, 3812.
[Fellers *et* Mellor 1986]	Fellers G. and Mellor M., 1986. *Concentration and Flux of Wind Blown Snow.* Technical report n° special report 86-11, Cold Regions Research and Engineering Laboratory.
[Föhn *et* Meister 1983]	Föhn P.M.B. and Meister R., 1983. Distribution of snow drifts on ridge slopes: measurements and theoretical approximations. *Annals of Glaciology*, **4**, 52–57.
[Frisch *et al.* 1986]	Frisch U., Hasslacher B. and Pomeau Y., 1986. Lattice-gas automata for the Navier-Stokes equation. *Phys. Rev. Lett.*, **56**, 1505.
[Frisch *et al.* 1987]	Frisch U., d'Humières D., Hasslacher B., Lallemand P., Pomeau Y. and Rivet J.-P., 1987. Lattice gas hydrodynamics in two and three dimension. *Complex Systems*, **1**, 649–707. Reprinted in *Lattice Gas Methods for Partial Differential Equations*, ed. G. Doolen, p.77, Addison-Wesley, 1990.
[Galam *et al.* 1998]	Galam S., Chopard B., Masselot A. and Droz M., 1998. Competing species dynamics: Qualitative advantage versus geography. *Eur. Phys. J. B*, **4**, 529–531.
[Gardner 1970]	Gardner M., 1970. The fantastic combinations of john conway's new solitaire game life. *Scientific American*, **220**(4), 120.
[Gauer 1999]	Gauer Peter, 1999. *Blowing and drifing snow in alpine terrain: a physically-based numerical model and related field measurements.* PhD Thesis, Swiss Federal Institute for Snow and Avalanche Research Davos.
[Hardy *et al.* 1973]	Hardy J., Pomeau Y. and de Pazzis O., 1973. Time evolution of a two-dimensional model system. I. Invariant states and time correlation functions. *J. Math. Phys.*, **14**, 1746.

[Hertig 1977] Hertig J.A., 1977. *Etude de la similitude entre ècoulements turbulents.* PhD Thesis, Ecole Polytechnique Fédérale de Lausanne. No 288.

[Hertig 1987] Hertig J.A., 1987. Protection contre les congères de la route nationale n9 au simplon. *Journal de la Construction*, **18**.

[Higuera *et al.* 1989a] Higuera F., Jimenez J. and Succi S., 1989. Boltzmann approach to lattice gas simulations. *Europhys. Lett*, **9**, 663.

[Higuera *et al.* 1989b] Higuera F., Jimenez J. and Succi S., 1989. Lattice gas dynamics with enhanced collision. *Europhys. Lett*, **9**, 345.

[Hillis 1989] Hillis W. Daniel, February 1989. Richard feynman and the connection machine. *Physics Today*, **42**(2), 78.

[Hockney 1996] Hockney Roger W., 1996. *The science of computer benchmarking.* Siam.

[Hou 1995] Hou S., 1995. *Lattice Boltzmann method for incompressible, viscous flow.* department of Mechanical Engineering, PhD Thesis, Kansas State University, Manhattan.

[Hou *et al.* 1996] Hou S., Sterling J., Chen S. and Doolen G.D., 1996. A lattice subgrid model for high Reynolds number flows. *Fields Institute Communications*, **6**, 151–166.

[Iversen 1980] Iversen J.D., 1980. Drifting snow similitude transport rate and roughness modeling. *Journal of Glaciology*, **26**(94), 393–403.

[Iversen 1981] Iversen J.D., 1981. Comparison of wind tunnel model and full-scale snow fence drifts. *Journal of Wind Engineering and Industrial Aerodynamics*, **8**, 231–249.

[Iversen 1982] Iversen J.D., 1982. Small scale modeling of snow drift phenomena. *Proc. Wind Tunnel Modeling for Civ. Eng. Appl.*, 421–434. Gaithersburg.

[Kadanoff 1986] Kadanoff Leo, September 1986. On two levels. *Physics Today*, **39**(9), 7–9.

[Kind *et* S.B. 1982] Kind R.J. and S.B. Murray, 1982. Saltation flow measurements relating to modelling of snowdrifting. *Journal of Wind Engineering and Industrial Aero.*, **10**, 89–102.

[Kind 1985] Kind R.J., 1985. *Snowdrifting: A Review of Modelling methods.* Technical report n° K1S5B6, Carlton University, Ottawa, Ontarion, Departement of Mechanical and Aeronautical Engineering.

[Knuth 1969] Knuth D.E., 1969. *The art of computer programming - Seminumerical algorithms.* Addison Wesley.

[Kobayashi 1969] Kobayashi S., 1969. Measurements of the wind drag force on the snow surface. *Low Temperature Science*, **Series A, Physical Sciences**(27), 87–97.

[Kobayashi 1972] Kobayashi D., 1972. Studies of snow transport in low-level drifting snow. *Contributions from the Institute of Low Temperature Science*, **Series A**(24), 1–58.

[Kumar *et al.* 1994] Kumar V., Grama A., Gupta A. and Karypis G., 1994. *Introduction to Parallel Computing: design and analysis of algorithm.* Benjamin/Cummings.

[Lancaster 1988] Lancaster N., 1988. Controls of eolian dune size and spacing. *Geology*, **16**, 972–975.

[Lee 1998] Lee S.J., 1998. Laboratory measurements of velocity turbulence field behind porous fences. *Jnl. of wind Engineering & Industrial aerodynamics.*

[Luthi 1998] Luthi Pascal O., 1998. *Lattice Wave Automata.* PhD Thesis, University of Geneva.

[Maeno *et al.* 1979] Maeno N., Araoka K., Nishimura K. and Kaneda Y., 1979. Physical aspects of the wind-snow interaction in blowing snow. *Journal of the faculty of Science, Hokkaido university*, **Series VII, VI**(1).

[Malevanets *et* Kapral 1998] Malevanets A. and Kapral R. Continuous-velocity lattice-gas models for fluid flow. 1998. preprint.

[Martin *et* chopard] Martin Marc and chopard Bastin, june. Low cost parallelising a way to be efficient. Vecpar'98, 1077–1088.

[Martinez 1996] Martinez H., february 1996. *Contribution à la modélisation du transport éolien de particules. Mesures de profiles de concentration en soufflerie diphasique.* PhD Thesis, Grenoble I.

[Masselot *et* Chopard 1998a] Masselot A. and Chopard B., 1998. A lattice Boltzmann model for particle transport and deposition. *Europhys. Lett.*, **42**, 259–264.

[Masselot *et* Chopard 1998b] Masselot A. and Chopard B., 1998. A lattice boltzmann model for snow transport and deposition. *Europhysics Letters*, **42**(3).

[Masselot 1995] Masselot A., March 1995. *A massively parallel approach to model snow transport by wind.* Technical report, University of Geneva.

[McNamara *et* Zanetti 1988] McNamara G.G. and Zanetti G., 1988. Use of the boltzmann equation to simulate lattice-gas automata. *Phys. Rev. Lett.*, **61**, 2332–2335.

[Mellor 1964] Mellor M., 1964. *Properties of snow.* Technical report n° III-A-1, CRREL US Army, Cold regions science and engineering.

[Mellor 1965] Mellor M., 1965. *blowing snow.* CRREL US Army.

[Metcalf *et* Reid 1999] Metcalf M. and Reid J., 1999. *Fortran 90/95 explained.* Oxford, second edition édition.

[Naaim *et* Brugnot 1992] Naaim F. and Brugnot G., 1992. *transport de la neige par le vent: connaissances de base et recommandations.* Technical report, Grenoble, France, CEMAGREF, Division Nivologie, Délégation aux risques majeurs. classeur 350 pp.

[Nalpanis 1985] Nalpanis P., 1985. Saltating and suspended oarticles over flat and sloping surfaces experiments and numerical simulations. *Proceedings of International Workshop on the physics of Blown Sand*, éd. par O.E. Eds Barndorff-Nielsen and al., Memories No 8, pp 37–66. Aarhus-Danmark.

[Nishimori *et* Ouchi 1993a] Nishimori H. and Ouchi N., 1993. Computational models for sand ripples and sand dune formation. *International Journal of Modern Physics*, **B 7**, 2025.

[Nishimori *et* Ouchi 1993b] Nishimori H. and Ouchi N., 1993. Formation of ripple patterns and dunes by wind-blown sand. *Physical Review Letters*, **71**, 197.

[Norem 1977] Norem H. Designing highways situated in areas of drifting snow. 1977. Draft translation 503 CRREL ADA028191. Hanover, New Hampshire.

[Ousland 1997] Ousland Børge, 1997. *Alone across Antarctica.* Cappelens Forlag A/S.

[Peng *et al.* 1999] Peng G., Xi H., Duncan C. and Chou S.-H., 1999. Finite volume scheme for the lattice boltzmann method on unstructured meshes. *Phys. rev. E*, **59**, To appear.

[Petkovic 1972] Petkovic S., 1972. *Modification des caract̄'eristiques d'une turbulence sous l'influence de particules solides en suspension.* PhD Thesis, USMG Grenoble.

[Pomeroy *et* Gray 1990] Pomeroy J.W. and Gray D.M., 1990. Saltation of snow. *Water Ressources Research*, **26(7)**, 1583–1594.

[Pomeroy 1988] Pomeroy J.W., 1988. *Wind Transport of Snow.* Saskatoon, PhD Thesis, University of Saskatchewan. 226 p.

[Qian *et al.* 1996a] Qian Y.H., Succi S. and Orszag S.A., 1996. Recent advances in lattice Boltzmann computing. *Annual Reviews of Computationnal Physics III*, éd. par Stauffer Dietrich. 195–242. World Scientific.

[Qian *et al.* 1996b] Qian Y.H., Succi S. and Orszag S.A., 1996. Recent advances in lattice boltzmann computing. *Annual Reviews of Computational Physics III*, éd. par Stauffer D. 195–242. World Scientific.

[Radok 1977] Radok U., 1977. Snow drift. *Journal of Glaciology*, **19**(81), 123–139.

[Raine *et* Stevenson 1977] Raine J.K. and Stevenson D.C., 1977. Wind protection model by fences in a simulated atmospheric boundary layer. *Journal of Industrial Aerodynamics*, **2**, 159–180.

[Raju *et al.* 1976] Raju R., Loeser J. and Plate E.J., 1976. velocity profiles and fence drag for a turbulent boundary layer along smooth and rough plates. *Journal of Fluid Mechanics*, **76**, 383–399.

[Raju *et al.* 1988] Raju R., Garde R.J., Singh S.K. and Singh N., 1988. Experimental study on characteristics of flow past porous fences. *Journal of Wind Eng. Ind. Aero.*, **29**, 155–163.

[Rey 1986] Rey L., 1986. *La neige, ses métamorphoses, les avalanches.* ANENA, Grenoble. 215 p.

[Rey *et al.* 1995] Rey P.A., Frachebourg L., Droz M. and Piasecki J., 1995. *Phys. Rev. Lett.*, **75**, 160.

[Rey *et al.* 1998] Rey P.A., Droz M. and Piasecki J., 1998. *Phys. Rev. E*, **57**, 138–145.

[Schmidt 1982] Schmidt R.A., 1982. properties of blowing snow. *Reviews of Geophysics and Space Physics*, **20**, 39–44.

[Schmidt *et al.* 1984] Schmidt R.A., Meister R. and Gubler H., 1984. Comparisons of snow drifting measurements at an alpine ridge crest. *Cold Regions Science and Technology*, **9**, 131–141.

[Seppala *et* Linde 1978] Seppala M. and Linde K., 1978. Wind tunnel studies of ripple formation. *Geog. Ann.*, **60**, 29–42.

[Sharp 1963] Sharp R.P., 1963. Wind ripples. *Journal of Geology*, **71**, 617–636.

[Sivardière *et* Castelle 1992] Sivardière F. and Castelle T., 1992. *Protection locale contre les accumlations de neige dues au vent: lutte contre la formation de corniches et la surcharge d'ouvrage paravalanches à Valneralp (Valais CH).* Technical report, EPFL-LASEN.

[Sivardière *et* Castelle 1993] Sivardière F. and Castelle T., 1993. protection locale contre les effets du transport de neige par le vent: les ouvrages à vent. *Symposium FEANI/DNDR, Lausanne.* Lausanne. 11 p.

[Sivardiere *et al.* 1993] Sivardière F., Castelle T. and Wuilloud C., 1993. protection locale contre les effets du transport de neige par le vent: l'exemple de valneralp. *Symposium FEANI/DNDR, Lausanne.* Lausanne. 8 p.

[Sumedley *et al.* 1993] Sumedley D.J., Kwok K.C.S. and Kim D.H., 1993. Snow drifting simulation around davis station workshop, antartica. *Journal of Wind Engineering*, n° 167, 153–162.

[Sumer *et* Deigaard 1981] Sumer B.M. and Deigaard R., 1981. Particle motions near the bottom in turblulent flow in an open channel, part 2. *Journal of Fluid Mechanic*, **109**, 311–347.

[Sumer *et* Fredsøe 1997] Sumer B. Multu and Fredsøe Jørgen, 1997. *Hydrodynamics around cylindrical structures.* World Scientific, *Advanced Series on Ocean Engineering*, volume 12.

[Sumer 1984] Sumer B.M., 1984. Lift forces on moving particles near boundaries. *Journal of Hydraulics Engineering*, **110-9**, 1272–1278.

[Sundsbø *et* Hansen 1996] Sundsbø P.-A. and Hansen E.W.M., 1996. Modelling and numerical simulation of snow drift around snow fences. *ICSE-3,Sendai.*

[Sundsbo 196] Sundsbø P.-A., 196. Computer simulations of snow drift around structures. *4th Symposium of building physics, Helsinki.*

[Sundsbo 1997]	Sundsbø P.-A., 1997. *Numerical modelling and simulation of snow drift.* PhD Thesis, Narvik Institute of Technology. ISBN 82-471-0047-9.
[Sutherland 1967]	Sutherland A.J., 1967. proposed mechanism for sediment entrainment by turbulent flows. *Journal of Geophys. Res.*, **72(24)**. 6183.
[Tabler 1980a]	Tabler R.D., 1980. Geometry and density of drifts formed by snow fences. *Journal of glaciology*, **26**(94), 405–419.
[Tabler 1980b]	Tabler R.D., 1980. Self-similarity of wind profiles in blowing snow allows outdoor modelling. *Journal of glaciology*, **26**(94), 421–434.
[Tabler 1991]	Tabler R.D., 1991. *Snow fence guide.* Technical report, Strategic highway research propgram, National Research Council, Washington DC. 61. pp.
[Takeuchi 1980]	Takeuchi M., 1980. Vertical profile and horizontal increase of drift snow transport. *Journal of Glaciology*, **26**(94), 481–492.
[Taneda 1979]	Taneda S., 1979. Visualization of separating stokes flows. *Journal of the physical society of Japan*, **46**(6), 1935–1942.
[Testori 1984]	Testori P., 1984. *Conditions de formation des congères au col du Simplon. Rapport interne de synthèse.* Technical report n° 517.114, Laboratoire IENER, EPF Lausanne.
[Tritton 1988]	Tritton D.J., 1988. *Physical fluid dynamics.* Clarendon Press.
[Trizac]	Trizac E. Private communication.
[Uematsu 1993]	Uematsu T., 1993. Numerical study on snow transport and drift formation. *Annals of Glaciology*, n° 18.
[Uematsu *et al.* 1991]	Uematsu T., Nakata T., Takeuchi K., Arisawa Y. and Kaenada Y., 1991. Three-dimensional numerical simulation of snow drift. *Cold Regions Science Technology*, **20**(1), 25–39.
[Vandewalle *et* Galam ress]	Vandewalle N. and Galam S., 1999, in press. Ripples versus dunes in a saltation-avalanche model. *Int. J. Mod. Phys. C.*

[Werner *et* Gillespie 1993] Werner B.Y. and Gillespie D.T., 1993. Fundamentally discrete stochastic model for wind ripple dynamics. *Physical Review Letters*, **71**, 3230.

[Werner *et al.* 1986] Werner B.T., Haff P.K., Livi R.P. and Anderson R.S., 1986. The measurement of eolian ripple cross-sectional shapes. *Geology*, **14**, 743–745.

[Yverniaux 1990] Yverniaux Ph., 1990. *Simulation eulero-lagrangienne du transport de particule solides en suspension dans un ècoulement turbulent en canal.* PhD Thesis, Thèse INP Grenoble Spécialité Mécanique.