

COMPTE RENDUS DU PROJET DE PHYSIQUE NUMÉRIQUE

Implémentation et étude de la méthode de Boltzmann sur réseau en FORTRAN

 Yohan Duarte^{1,*}
¹Étudiant en M1 PFA à Sorbonne Université, aka Pacidus

*Me contacter. pacidus@gmail.com

Abstract

La résolution des équations Navier-Stokes constituent aujourd’hui un enjeu majeur dans de nombreux domaines. Dans le cadre du projet de L’UE de Physique numérique, nous avons eu l’occasion d’implémenter la méthode de Boltzmann sur réseau. Cette méthode de résolution des équations de Navier-Stokes constitue un changement de paradigme par rapport aux méthodes plus classiques.

Mots clés: Navier-Stokes, méthode lattice Boltzmann, Bhatnagar-Gross-Krook, dynamique des fluides, Projet Numérique

Introduction

Des plasmas qui composent les nébuleuses aux arrivées d’eau de nos maisons en passant par les systèmes de refroidissement des centrales nucléaires et le flux sanguin alimentant notre métabolisme. Les fluides composent la majorité de la matière visible dans notre univers [1]. La dynamique de ces fluides sont décrits par une équation rédigée au milieu XIXe siècle [2]. Fruit de la collaboration de générations de physiciens, les équations de Navier-Stokes sont toujours aujourd’hui au cœur de nombreuses problématiques. La question de l’existence et de la régularité des solutions des équations de Navier-Stokes en 3D constitue encore aujourd’hui un problème ouvert et est l’un des problèmes du prix du millénaire de l’institut de mathématique de Clay [3].

À défaut d’une solution analytique, c’est dès la première moitié du XXe siècle que les physiciens développent les premières méthodes de résolutions numériques des équations de Navier-Stokes [4]. Depuis l’apparition des premiers ordinateurs, on a assisté à l’émergence de nombreuses méthodes numériques de résolution de dynamique des fluides. La branche de la physique utilisant et étudiant ces méthodes et nommée *Computational fluid dynamic* aussi abrégée en CFD. La méthode qui nous intéresse ce nomme la méthode de Boltzmann sur réseau plus couramment abrégée par LBM pour *Lattice Boltzmann Methods*.

Les LBM apparaissent au milieu des années 1980 [5, 6, 7, 8] depuis lors cette méthode gagne en popularité comme l’illustre les figures¹ 1, 2 et 3.

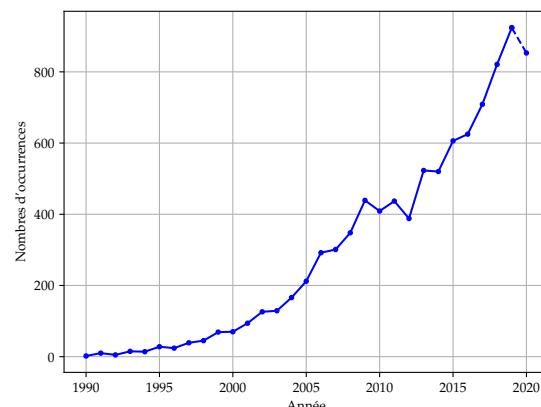


FIGURE 1. Évolution du nombre de publication référencée dans Google Scholar possédant l’occurrence «Lattice Boltzmann Methods» au cours de ces 30 dernières années.

1. La méthodologie permettent d’obtenir ces valeurs est critiquable néanmoins elles montrent une tendance générale.

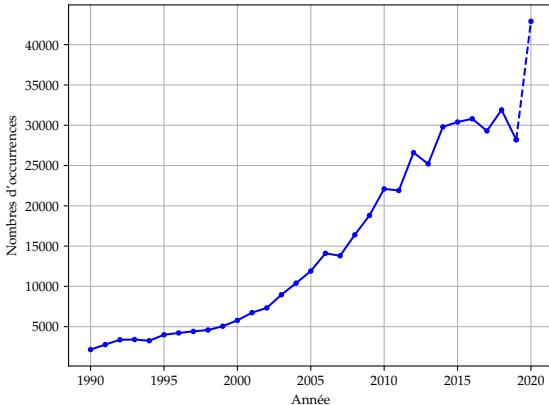


FIGURE 2. Évolution du nombre de publication référencée dans Google Scholar possédant l'occurrence «Computational fluid dynamics» au cours de ces 30 dernières années.

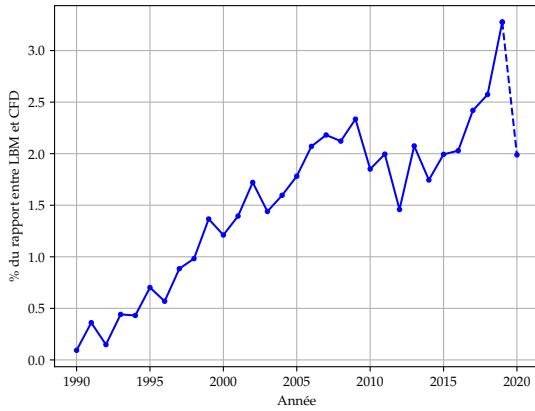


FIGURE 3. Rapport en pourcentage entre la figure 1 et la figure 2.

Ce succès peut s'expliquer par le fait que les LBM profitent de différentes propriétés intéressantes :

- Polyvalence :

Les LBM peuvent être utilisées sur des problèmes très différents (aérodynamiques, médicaux, ...).

- Complexité :

Les LBM peuvent facilement simuler des objets à géométries complexes et peuvent inclure des interactions entre plusieurs fluides ainsi que de la thermodynamique et d'autres forces en tout genres.

- Parallélisation :

Les LBM profitent de pleinement des accélérations permises par les architectures multicores et GPU les simulations peuvent donc être déployées sur différents HPC et clusters de calculs.

- Pas d'inconnues :

Les LBM contrairement à d'autres méthodes plus communes résolvant directement Navier-Stokes possède

toutes les quantités nécessaires à la résolution déjà définies² et ce de manière locale³.

Il est à noter que ces qualités sont inhérentes aux LBM et ne demandent pas de modifications majeures pour être implémentées⁴.

Fluides, Macro, Micro et Méso

Les fluides peuvent être modéliser à plusieurs échelles, chacune des ces descriptions utilise des approches différentes et des outils mathématiques propres. La description continue décrit le fluide comme étant une série de champ remplissant l'espace et est régie par les équations de Navier-Stokes ici dans le cas d'un fluide incompressible⁵ :

- Bilan de la quantité de mouvement :

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \vec{\nabla}) \vec{u} = -\frac{\vec{\nabla} P}{\rho} + \nu \Delta \vec{u} + \frac{\vec{F}_{\text{ext}}}{\rho}, \quad (1)$$

- condition sur le champ de vitesse pour un fluide incompressible :

$$\vec{\nabla} \cdot \vec{u} = 0. \quad (2)$$

Cette description est la description macroscopique du fluide elle se place à une échelle dans laquelle les quantités varient continûment dans l'espace et permettent ainsi l'utilisation de l'analyse mathématique. Cette description est suffisante pour la plupart des applications, à tel point que l'on résume souvent la mécanique des fluides à sa description continue⁶.

On peut aussi décrire les fluides comme une collection de particules, atomes et molécules, qui interagissent. Décrire un fluide sur la base de leurs interactions constitue les prémisses des LBM, en effet, la méthode pré-dépassant les LBM (la méthode LGA pour lattice gas automata) consistait en un réseau dans lequel pouvais interagir des particules, à chaque instant une maille du réseau était soit vide ou pleine.

Cette méthode est très laborieuse et possède beaucoup d'artefacts dépendant du type de maillage utilisée [9].

Une dernière échelle permettant la description d'un fluide et la description mésoscopique. A cette échelle intermédiaire on décrit le fluide en terme de densité de probabilité $f(\vec{r}, \vec{v}, t)$ de trouver une particule à une position dans l'espace \vec{r} , et un temps t , avec une vitesse \vec{v} . Cette échelle intermédiaire de description et connue sous le nom de théorie cinétique des gaz. Le pendant mésoscopique des équations de Navier-Stokes est l'équation de Boltzmann

$$\frac{\partial f}{\partial t} + \vec{v} \cdot \vec{\nabla}_{\vec{r}} f + \frac{\vec{F}_{\text{ext}}}{\rho} \cdot \vec{\nabla}_{\vec{v}} f = C(f). \quad (3)$$

Il est possible de d'exprimer les équations de Navier-Stokes comme une petite perturbation de la distribution de Maxwell-Boltzmann en utilisant le méthode de Chapman-Enskog [10]. Cette dernière échelle de description est celle utilisée par les LBM et est à l'origine des propriétés uniques des LBM.

2. Elle ne demande donc pas (par exemple) de résoudre les équations de poisson pour la pression à chaque itération.

3. Ce qui explique le fait que les LBM sont hautement parallélisable.

4. À comprendre que l'algorithme de résolution reste le même.

5. Ce qui correspond par exemple à l'eau et à l'air à température et pression usuelles.

6. Cette forme d'essentialisme et parfaitement illustré dans ce compte-rendu.

Grandeurs & Conventions

Nous allons nous attarder ici⁷ sur les différentes grandeurs nécessaire à la description d'un fluide ainsi que les conventions qui seront utilisées dans la suite du compte-rendu⁸.

Commençons par Les grandeurs macroscopiques, elles sont les suivantes :

- Le champ de vitesse \vec{u} du fluide :

$$[\vec{u}] = L \cdot T^{-1},$$

$$\vec{u}(\vec{r}, t) : \mathbb{R}^{D+1} \rightarrow \mathbb{R}^D.$$

Ici D indique la dimension spatiale de notre problème pour la suite nous nous limiterons au cas 2D ($D=2$).

- La pression interne au fluide P :

$$[P] = M \cdot L^{-1} \cdot T^{-2},$$

$$P(\vec{r}, t) : \mathbb{R}^{D+1} \rightarrow \mathbb{R}.$$

La pression est l'une des formes que prend l'énergie interne du fluide elle peut se convertir en énergie cinétique et réciproquement.

- La masse volumique du fluide ρ :

$$[\rho] = M \cdot L^{-3},$$

$$\rho(\vec{r}, t) : \mathbb{R}^{D+1} \rightarrow \mathbb{R}^+.$$

Pour un fluide incompressible, ρ est une constante du fluide et ne varie pas dans le temps néanmoins, pour la suite, il reste utile de le définir comme une fonction de l'espace et du temps.

- La viscosité cinématique du fluide ν :

$$[\nu] = L^2 \cdot T^{-1},$$

$$\nu \in \mathbb{R}_*^+.$$

La viscosité cinématique est une constante du fluide et peut se comprendre comme la capacité du fluide «diffuser» sa vitesse.

- Le champ de force externe \vec{F}_{ext} :

$$[\vec{F}_{\text{ext}}] = M \cdot L \cdot T^{-2},$$

$$\vec{F}_{\text{ext}} = 0.$$

Dans la suite du compte-rendu nous nous limiterons à un champ de force nul.

- Le nombre de Reynolds Re :

$$[\text{Re}] = \emptyset,$$

$$\text{Re} \in \mathbb{R}_*^+.$$

Le nombre de Reynolds est un nombre qui permet de caractériser un écoulement. Si deux fluides différents sont étudiés dans la même géométrie et avec la même valeur de Re l'écoulement est le même.

7. Les auteurs souhaitent ici préserver la santé mentale du lecteur.

8. Il existe autant de conventions différentes que de papier sur les LBM nous nous permettrons donc de rajouter notre pierre à cette horrible tradition.

Grandeurs mésoscopiques apparaissent dans la définition des LBM et sont les suivantes :

- La vitesse des particules \vec{v} :

$$[\vec{v}] = L \cdot T^{-1},$$

$$\vec{v} \in \mathbb{R}^D$$

Vitesse des particules à différentier de \vec{u} la vitesse macroscopique.

- La densité de probabilité f :

$$[f] = \emptyset,$$

$$f(\vec{r}, \vec{v}, t) : \mathbb{R}^{2D+1} \rightarrow \mathbb{R}^+.$$

La densité de probabilité ne possède pas de dimensions néanmoins intégré sur un volume de l'espace des phase elle est proportionnelle à la densité et donc à la masse volumique ρ .

- L'opérateur collision $C(f)$:

$$[C(f)] = \emptyset,$$

$$C(f) : \mathbb{R}^+ \rightarrow \mathbb{R}^+.$$

L'opérateur collision est une fonction qui décrit l'interaction des particules entre elles, dans le cas des LBM l'opérateur le plus courant (et celui que nous allons utiliser) est l'opérateur de Bhatnagar-Gross-Krook (abrégé en BGK⁹).

- La densité de probabilité à l'équilibre f^{eq} : correspond à la densité de probabilité à l'équilibre étant donné les grandeurs macroscopiques \vec{u}, ρ . Il apparaît dans l'opérateur collision BGK.

- La vitesse du son dans le fluide c_s :

$$[c_s] = L \cdot T^{-1},$$

$$c_s \in \mathbb{R}_*^+.$$

Vitesse du son dans le fluide (c'est aussi la vitesse moyenne des particules). Il apparaît dans le calcul de f^{eq} .

- coefficient de relaxation τ :

$$[\tau] = \emptyset,$$

$$\tau \in \left[\frac{1}{2}, +\infty \right[.$$

Dépend de ν, c_s et du pas de temps il apparaît dans l'opérateur collision BGK.

9. C'est pour cela que certains auteurs nomment les LBM basées sur cet opérateur des LBGK.

Pour passer de l'équation de Boltzmann aux LBM il va falloir discréteriser certaines grandeurs, voici les grandeurs discréterisées pour la simulation :

- Le pas de discréterisation spatial δ_l :

$$[\delta_l] = L$$

$$\delta_l = \mathbb{R}_*^+.$$

On discréterise les valeurs que peuvent prendre \vec{r} .

- Le pas de discréterisation temporel δ_t :

$$[\delta_t] = T$$

$$\delta_t = \mathbb{R}_*^+.$$

On discréterise les valeurs que peuvent prendre t .

- Le jeu de vitesses \vec{e}_i :

$$[\vec{e}_i] = L \cdot T^{-1}$$

$$\vec{e}_i \in \mathbb{R}.$$

On discréterise les valeurs que peuvent prendre \vec{v} au total on choisira Q vitesses. Toutes les LBM sont définies par leur dimension nombre spatiales D et leur nombre de vitesses microscopiques Q .

- La densité de probabilité discréterisée f_i :

$$[f_i] = \emptyset,$$

$$\dim(f_i) = D + 1.$$

La densité de probabilité et désormais discréterisée dans l'espace des phase, c'est donc un tableau de taille finie, qui dépend de notre simulation. f^{eq} est discréterisé de la même manière.

- La vitesse de maille c :

$$[c] = L \cdot T^{-1},$$

$$c \in \mathbb{R}_*^+.$$

Correspond à la vitesse nécessaire pour ce déplacer d'une maille à une autre en un temps δ_t .

- La densité ρ :

$$[\rho] = \emptyset,$$

$$\rho : \mathbb{R}^D \rightarrow \mathbb{R}^+.$$

ρ est désormais adimensionné et vaut la densité du fluide en chaque point.

- La probabilité à l'équilibre w_i :

$$[w_i] = \emptyset,$$

$$w_i \in \mathbb{R}_*^+.$$

w_i correspond à la probabilité associée à une vitesse \vec{e}_i pour une vitesse macroscopique $\vec{u} = \vec{0}$.

Discréterisations pour le cas D2Q9

Comme dit plus tôt, pour pouvoir passer de l'équation de Boltzmann aux LBM¹⁰ il faut discréteriser l'espace des phases. Dans le volume d'abord : \vec{r} est borné à notre espace d'intérêt et on échantillonne cet espace avec un pas de longueur δ_l . Puis l'on vas échantillonner nos vitesses dans notre cas on veux que pour chaque instant δ_t des particules puissent s'échanger avec les deux plus proches voisins d'un nœud de notre réseau. Afin de préserver l'isotropie des vitesses \vec{v} nous ne privilégierons aucune direction en échantillonnant de la sorte on obtiens le jeu de vitesses représenté sur la figure 4.

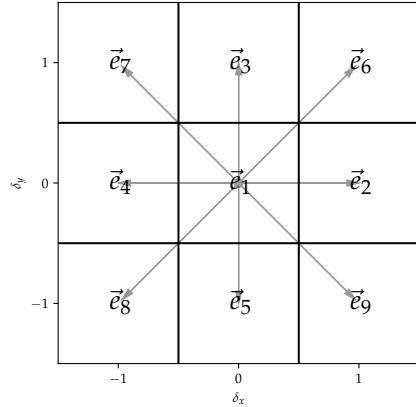


FIGURE 4. Vitesses \vec{e}_i du réseau pour D2Q9.

Il faut bien comprendre que nous aurions pu définir n'importe quel jeu de vitesses \vec{e}_i arbitraire¹¹, tout comme on a choisi un jeu de positions \vec{r} dans lequel notre simulation ce déroulera, cette discréterisation dans l'espace des phases n'est fondamentalement pas beaucoup différent de ce que l'on fait plus couramment.

Étant donnée la construction de notre réseau on peux définir la vitesse de maille c :

$$c := \frac{\delta_l}{\delta_t}. \quad (4)$$

A l'aide de la figure 4 et de la définition de c on peux définir les valeurs de \vec{e}_i pour notre simulation :

$$\vec{e}_i := c \begin{bmatrix} 0 & 1 & 0 & -1 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 1 & 0 & -1 & 1 & 1 & -1 & -1 \end{bmatrix}. \quad (5)$$

En utilisant la distribution de Maxwell-Boltzmann il nous est possible de calculer les poids w_i de la distribution à l'équilibre (et sans vitesse \vec{u}). Pour une LBM D2Q9 les poids w_i sont les suivants :

$$w_i := \left[\frac{4}{9}; \frac{1}{9}; \frac{1}{9}; \frac{1}{9}; \frac{1}{36}; \frac{1}{36}; \frac{1}{36}; \frac{1}{36}; \frac{1}{36} \right], \quad (6)$$

avec cette distribution a bien $\sum_{i=1}^9 w_i = 1$.

Précédemment nous avons dis que c_s la vitesse du son était la vitesse moyenne $\langle ||\vec{v}|| \rangle$ de nos particules. On peut donc écrire :

10. Pour plus de précisions vous pouvez lire l'article [11].

11. À partir du moment où l'on respecte l'isotropie.

$$c_s^2 \delta_{\alpha\beta} = \sum_{i=1}^Q w_i e_{i\alpha} e_{i\beta}, \quad (7)$$

appliqué au cas 2DQ9 cela nous donne :

$$c_s = \frac{c}{\sqrt{3}}. \quad (8)$$

Nous avons ainsi définie toutes les grandeurs dépendent de notre choix de D2Q9.

De l'équation de Boltzmann aux LBM (LBGK)

Dans le cas que nous avons choisis nous pouvons réécrire l'équation de Boltzmann (3) de la manière suivante :

$$\frac{\partial f}{\partial t} + \vec{v} \cdot \vec{\nabla}_{\vec{r}} f = C(f), \quad (9)$$

si l'on remplace $C(f)$ par l'opérateur BGK :

$$C(f) = -\frac{1}{\tau} (f - f^{\text{eq}}), \quad (10)$$

on obtiens :

$$\frac{\partial f}{\partial t} + \vec{v} \cdot \vec{\nabla}_{\vec{r}} f = -\frac{1}{\tau} (f - f^{\text{eq}}). \quad (11)$$

Si l'on ajoute f des deux cotées et que l'on discréte l'équation 11 on obtiens notre résultat final :

$$f_i(\vec{r} + \delta_t \vec{e}_i, t + \delta_t) = f_i - \frac{1}{\tau} (f_i - f_i^{\text{eq}}). \quad (12)$$

Reste encore à définir les termes apparaissant dans notre équation,

• τ :

$$\tau = \frac{\nu}{c_s^2 \delta_t} + \frac{1}{2}, \quad (13)$$

• f_i^{eq} :

$$f_i^{\text{eq}} = w_i \rho \left(1 + \frac{\vec{u} \cdot \vec{e}_i}{c_s^2} + \frac{(\vec{u} \cdot \vec{e}_i)^2}{2c_s^4} - \frac{\vec{u} \cdot \vec{u}}{2c_s^2} \right), \quad (14)$$

• ρ :

$$\rho = \sum_{i=1}^Q f_i, \quad (15)$$

• \vec{u} :

$$\vec{u} = \frac{1}{\rho} \sum_{i=1}^Q f_i \vec{e}_i. \quad (16)$$

Conditions aux bornes & interactions

Nos équations décrivent notre fluide dans le domaine que nous avons défini mais ce n'est pas suffisant, il reste encore à définir comment notre fluide ce comporte aux frontières de notre domaine et comment il interagit avec notre domaine.

En mécanique des fluides numérique il y as un jeu de condition aux bornes qui suffisent à décrire la majorité des situations nous allons présenter ici celles que nous utiliserons dans nos simulations :

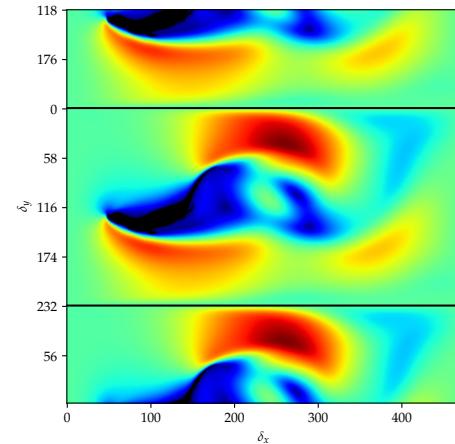


FIGURE 5. Exemple de condition périodique matérialisée par les droites noires.

Conditions périodiques :

Acclamé par la critique, les conditions périodiques aux bornes possèdent le bon goût d'être simples à implémenter et de ne présenter quasiment pas d'effet secondaire¹².

Dans ce cas il suffit de définir que les f_i franchissant la frontière «s'échangent» avec leurs équivalents de la frontière rattaché, dans le cas de la figure 5 et avec les indices de la figure 4 on peut réécrire cela de manière plus formelle :

$$\begin{aligned} \forall (\delta_x + \text{sign}(e_{i,x})) \in \Omega \\ i \in [7, 3, 6] & \quad f_i(\delta_x, 0) \rightarrow f_i(\delta_x + \text{sign}(e_{i,x}), 232), \\ i \in [8, 5, 9] & \quad f_i(\delta_x, 232) \rightarrow f_i(\delta_x + \text{sign}(e_{i,x}), 0), \end{aligned} \quad (17)$$

avec Ω l'ensemble de la frontière considérée et $\text{sign}()$ la fonction qui renvoie le signe.

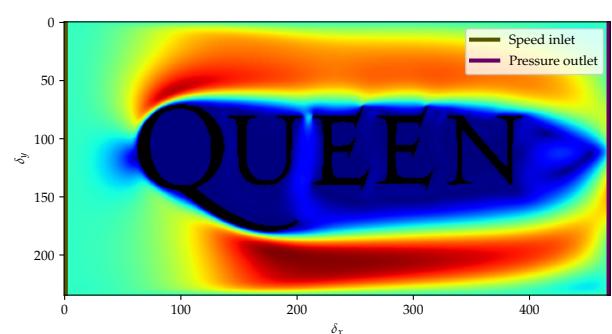


FIGURE 6. Exemple de conditions d'entrée/sorties matérialisée par deux droites respectivement à gauche et à droite du graphique.

12. Si ce n'est se placer dans un plan infini...

Entrée à vitesse imposée :

Quand il s'agit des entrées et de sorties il existe une référence dans le domaine et c'est l'article de Qisu Zou et Xiaoyi He¹³ [12]. Dans le cas de notre implémentation il nous a fallu les redémontre dans le cadre des équations dimensionnées.

L'idée est la suivante : Au niveau de l'entrée¹⁴ dans le cas d'une entrée comme présentée dans la figure 6 après la diffusion f_3, f_7, f_4, f_8, f_5 sont connus. On vas donc utiliser on les équations 15 et 16 pour déterminer les inconnues f_6, f_2, f_9 et ρ , si l'on écrit explicitement les sommes des équations 15 et 16 pour notre simulation D2Q9 on obtiens :

$$\begin{aligned}\rho &= f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9, \\ \rho u_x &= c(f_2 - f_4 + f_6 - f_7 - f_8 + f_9), \\ \rho u_y &= c(f_3 - f_5 + f_6 + f_7 - f_8 - f_9).\end{aligned}$$

On peut réécrire ces équations pour faire apparaître les f_i inconnue d'un coté et les autres termes de l'autre :

$$f_6 + f_2 + f_9 = \rho - (f_1 + f_3 + f_4 + f_5 + f_7 + f_8), \quad (18)$$

$$f_6 + f_2 + f_9 = \frac{\rho u_x}{c} + (f_4 + f_7 + f_8), \quad (19)$$

$$f_6 - f_9 = \frac{\rho u_y}{c} - (f_3 - f_5 + f_7 - f_8). \quad (20)$$

En soustrayant les équations 18 et 19 et en réarrangeant les termes on obtiens :

$$\rho = \frac{c}{c - u_x} (f_1 + f_3 + f_5 + 2(f_4 + f_7 + f_8)), \quad (21)$$

étant donnée que l'on impose la vitesse \vec{u} sur l'inlet l'équation 21 nous donne la valeur de l'inconnue ρ . Pour poursuivre le raisonnement il faut faire l'hypothèse¹⁵ que $f_2 - f_2^{\text{eq}} = f_4 - f_4^{\text{eq}}$. Si l'on développe avec l'équation 14 et 8 cette hypothèse et que l'on simplifie on obtiens :

$$f_2 = f_4 + \frac{2\rho u_x}{3c}. \quad (22)$$

Si l'on remplace 22 dans 19 on obtiens :

$$f_6 + f_9 = \frac{\rho u_x}{3c} + f_7 + f_8, \quad (23)$$

en sommant l'équation 23 avec 20 et en simplifiant on obtiens :

$$f_6 = f_8 + \frac{\rho}{2c} \left(\frac{u_x}{3} + u_y \right) + \frac{1}{2}(f_5 - f_3), \quad (24)$$

la même chose en soustrayant l'équation 23 avec 20 et en simplifiant on obtiens :

$$f_9 = f_7 + \frac{\rho}{2c} \left(\frac{u_x}{3} - u_y \right) + \frac{1}{2}(f_3 - f_5), \quad (25)$$

dans le cas d'un écoulement incompressible on peut prendre $\rho = 1$ ce qui permet encore de simplifier nos équations. Nous avons donc notre jeux d'équations suivant pour notre entrée :

$$\begin{aligned}f_2 &= f_4 + 2\frac{u_x}{3c}, \\ f_6 &= f_8 + \frac{1}{2} \left(\frac{u_x}{3c} + \frac{u_y}{c} + f_5 - f_3 \right), \\ f_9 &= f_7 + \frac{1}{2} \left(\frac{u_x}{3c} - \frac{u_y}{c} + f_3 - f_5 \right).\end{aligned} \quad (26)$$

13. souvent abrégée en Zou & He

14. Inlet en anglais (et dans la littérature en général).

15. C'est là où ce trouve le «tour de force» de l'article [12]

Sortie à pression imposée :

Pour la sortie on suppose que $u_y = 0$ et que $\rho = \rho_{\text{out}}$, il nous reste à trouver les valeurs de u_x ainsi que de f_7, f_4, f_8 (dans le cas de la figure 6). Avec le même raisonnement que précédemment et avec 15 et 16 on obtiens :

$$f_7 + f_4 + f_8 = \rho_{\text{out}} - (f_1 + f_2 + f_3 + f_5 + f_6 + f_9), \quad (27)$$

$$f_7 + f_4 + f_8 = \frac{\rho_{\text{out}} u_x}{c} + (f_2 + f_6 + f_9), \quad (28)$$

$$f_7 - f_8 = -f_3 + f_5 - f_6 + f_9. \quad (29)$$

En soustrayant les équations 27 et 28 et en réarrangeant les termes on obtiens :

$$u_x = c - \frac{c}{\rho_{\text{out}}} (f_1 + f_3 + f_5 + 2(f_6 + f_2 + f_9)), \quad (30)$$

étant donnée que l'on impose la densité ρ_{out} sur l'outlet l'équation 30 nous donne la valeur de l'inconnue u_x . En réutilisant la même hypothèse et la même méthode que précédemment on obtiens :

$$f_4 = f_2 + \frac{2\rho_{\text{out}} u_x}{3c}, \quad (31)$$

$$f_7 = f_9 + \frac{1}{2} \left(\frac{\rho_{\text{out}} u_x}{3c} + f_5 - f_3 \right), \quad (32)$$

$$f_8 = f_6 + \frac{1}{2} \left(\frac{\rho_{\text{out}} u_x}{3c} + f_3 - f_5 \right). \quad (33)$$

La encore dans le cas d'un écoulement incompressible on peut prendre $\rho_{\text{out}} = 1$ ce qui permet de simplifier nos équations. Nous avons donc notre deuxième jeux d'équations suivant pour notre sortie :

$$\begin{aligned}u_x &= c(1 - [f_1 + f_3 + f_5 + 2(f_6 + f_2 + f_9)]), \\ f_4 &= f_2 + \frac{2u_x}{3c}, \\ f_7 &= f_9 + \frac{1}{2} \left(\frac{u_x}{3c} + f_5 - f_3 \right), \\ f_8 &= f_6 + \frac{1}{2} \left(\frac{u_x}{3c} + f_3 - f_5 \right).\end{aligned} \quad (34)$$

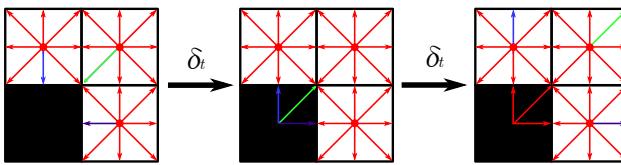


FIGURE 7. Illustration du processus de collision avec un objet (en noir).

collision sans glissement :

Avec tout ce dont nous venons de discuter jusqu'ici il est évident que nous allons étudier un flux¹⁶, mais pour l'instant nous n'avons pas encore discuté de l'interaction entre notre fluide et autre chose que lui-même. Les collisions sans glissement¹⁷ peuvent s'interpréter de la manière suivante : étant donnée une population de particules f_i associée à une vitesse microscopique \vec{e}_i qui rentre en collision avec un solide, elle va rebondir et repartir avec une vitesse \vec{e}'_i opposée. Autrement dit pour simuler une collision il suffit d'échanger les populations f_i de particules entre les vitesses correspondante. Si l'on se réfère à la figure 4 nous avons les échanges suivants :

$$\begin{aligned} f_2 &\rightleftharpoons; f_4, \\ f_3 &\rightleftharpoons; f_5, \\ f_6 &\rightleftharpoons; f_8, \\ f_7 &\rightleftharpoons; f_9, \end{aligned} \quad (35)$$

f_1 correspondant aux particules au repos n'est pas échangé.

Algorithme & implémentation

Il est désormais temps d'implémenter notre LBM, l'algorithme ce décompose en trois étapes :

- Initialisation :

On définit les valeurs aux bornes et l'on applique les rebond sur les objets (on échange les populations).

- Flux :

On déplace les populations de $\vec{e}_i \delta t$.

- Collision :

On applique l'opérateur collision sur les populations f_i .

On répète ces étapes en boucle pour simuler le comportement du fluide.

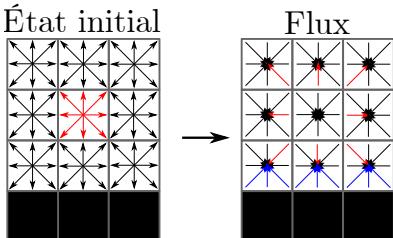


FIGURE 8. Schéma des différentes étapes de résolution de la LBM.

Cette structure est effectivement bien visible dans notre code FORTRAN :

```

!=====
!                                         The main program.
!=====

Program LBM2D
  Use Var
  Use Dconst, Only: L, H, N
  Implicit None
  Integer(4) :: dN, d = 100, ni = 0
  Call Allocatall                         ! Alloue les variables en mémoire
  Call InitF                               ! Initialise les variables

  Do dN = 0, N
    If (Modulo(dN, d) == 0) Then
      Call Savefile(ni)                   ! Sauvegarde les valeurs intermédiaires pour
      ni = ni + 1
    End If

    CALL IOlet                            ! Initialisation
    CALL Bound                            ! Initialisation
    CALL Stream                           ! Flux
    CALL CMacro                           ! Collision : calcul les valeurs macro
    CALL CFeq                             ! Collision : calcul les valeurs de feq
    CALL Collide                          ! Collision : applique l'opérateur collisi
  End Do

  Call Deallocaall                      ! Libère la mémoire
End Program LBM2D

```

durant l'implémentation nous avons fait le choix d'utiliser des sousroutines afin de minimiser l'impact mémoire (La gestion de la mémoire est un problème critique dans notre code nous avons donc privilégié le passage par référence plutôt que le passage par valeur). De plus pour accélérer notre code nous avons utilisée énormément de boucles Do en respectant l'ordre des indices, comme par exemple dans la subroutine collide :

```

!=====
!                                         Collide Step
!=====

Subroutine Collide
  Use Var, Only: F, Feq
  Use Cconst, Only: itau, mitau
  Use Dconst, Only: L, H
  Use Lconst, Only: Q
  Implicit None
  Integer(4) :: i, j
  Integer(1) :: k

  Do j = 1, H
    Do i = 1, L
      Do k = 1, Q
        F(k,i,j) = mitau*F(k,i,j) + itau*Feq(k,i,j)
      End Do
    End Do
  End Do
End Subroutine

```

cette subroutine est l'implémentation de l'équation 12.

16. Flow Driven Analyses dans la littérature.

17. Aussi appelées «no slip boundaries».

Premiers tests et écoulements indépendant du temps.

Notre étude ce portera dans un premier temps à vérifier que notre implémentation de LBM concorde avec la théorie. Pour cela nous allons étudier l'écoulement de Poiseuille nous allons donc placer un profil en forme de tube dans notre simulation. Pour vérifier la corrélation avec les résultats théorique nous allons intégrer le débit volumique à deux niveau et on va pouvoir ainsi vérifier la conservation du débit volumique. En plus nous avons la formule théorique de l'écoulement de Poiseuille

$$v(z) = v_{\max} \left(\frac{4z}{h} - \frac{4z^2}{h^2} \right), \quad (36)$$

en intégrant sur h on obtiens v_{\max} en fonction du débit volumique D_v :

$$D_v = v_{\max} \frac{2h}{3}. \quad (37)$$

Étant donné que notre écoulement est incompressible en intégrant numériquement sur une première section on obtiens donc le profil de vitesse pour la deuxième section :

$$v(x) = D_v \frac{3}{2h} \left(\frac{4z}{h} - \frac{4z^2}{h^2} \right). \quad (38)$$

Voici les différents résultats obtenus :

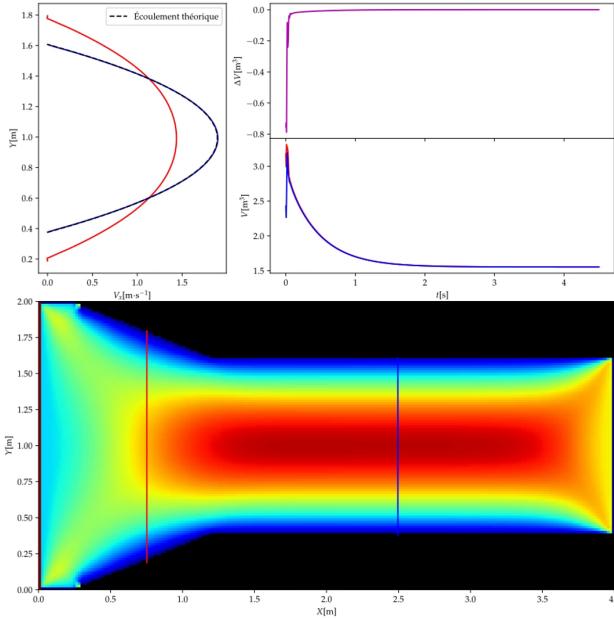


FIGURE 9. Étude de l'écoulement de Poiseuille dans un cas simple

On observe que le débit volumique est conservée (à l'équilibre) et plus le profil des vitesses simulé colle parfaitement à la théorie.

On observe pour la figure 10 une nouvelle fois un très bon accord avec la théorie le débit volumique est toujours conservé. À partir de la figure 11 le profil des vitesses ne colle plus totalement à la théorie, rien de très inquiétant étant donné que ce profile sort du domaine de définition de la théorie de poiseuille néanmoins le débit volumique est toujours conservé.

Maintenant que nous avons confirmé que notre simulation concordais avec la théorie, nous pouvons essayer de simuler des

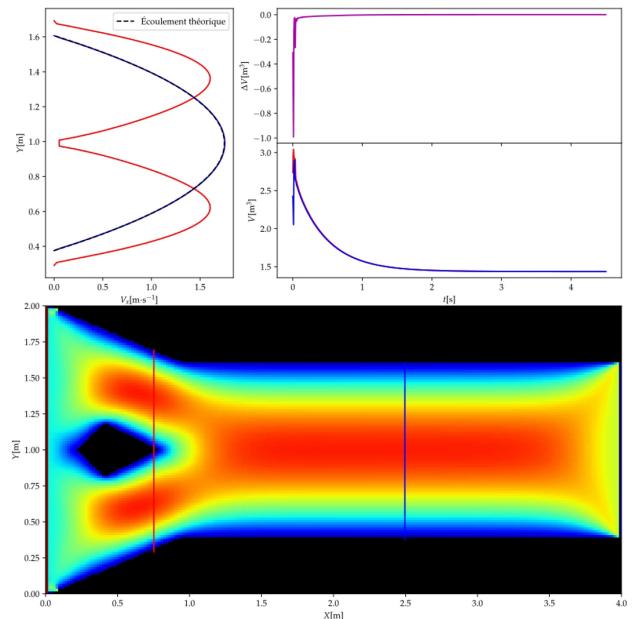


FIGURE 10. Étude de l'écoulement de Poiseuille dans une géométrie plus complexe

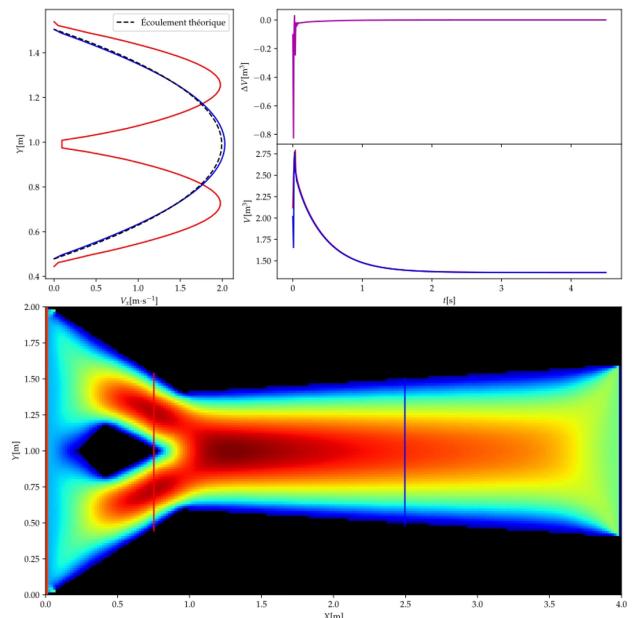


FIGURE 11. Étude de l'écoulement de Poiseuille dans une géométrie encore un peu plus complexe

systèmes non solvables analytiquement comme pour la figure 12.

La méthode lattice Boltzmann s'adapte donc bien à des géométries variées, et le débit volumique est toujours conservé. Mais jusqu'à présent nous n'avons fait qu'étudier des flux qui converge vers des solutions indépendantes du temps.

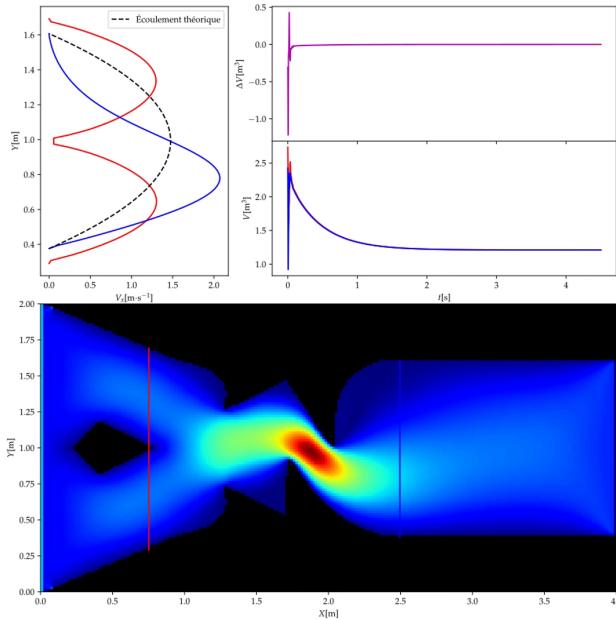


FIGURE 12. Étude de l'écoulement dans une géométrie complexe

Solutions périodiques et allée de tourbillons de Von Karman.

Dans le but de pousser notre implémentation de la LBM un peu plus loin, nous avons simulée le flux autour d'une aile d'avion, le but était d'obtenir nos premiers écoulements dépendant du temps, voici en figure 13 un des résultat que nous avons obtenu.

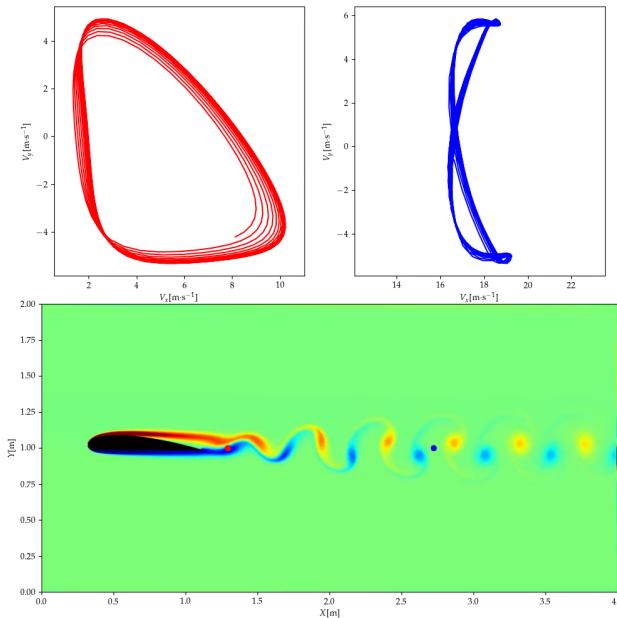


FIGURE 13. Allée de Von Karman pour $Re \approx 400$

Afin d'étudier plus quantitativement nos résultats nous avons choisi deux points d'intérêt au niveau de l'allée de Von

Karman et nous avons tracé la trajectoire dans l'espace des phases de ces points, passée un régime de transition, ces trajectoires convergeaient vers une orbite périodique. L'analyse de la trajectoire dans l'espace des phases est un outil très puissant quand il es question de caractériser les écoulements auquel nous avons à faire. Très vite lors de nos simulations nous avons systématiquement utilisé le nombre de Reynolds afin de pouvoir avoir une idée avant de lancer la simulation de la physique que nous allions avoir.

Solution chaotiques

Dans un derniers effort nous avons poussé notre simulation vers les haut nombres de Reynolds, nous avons pu observer des comportements chaotiques. Là encore nos visualisation nous on permis de quantifier ce comportement.

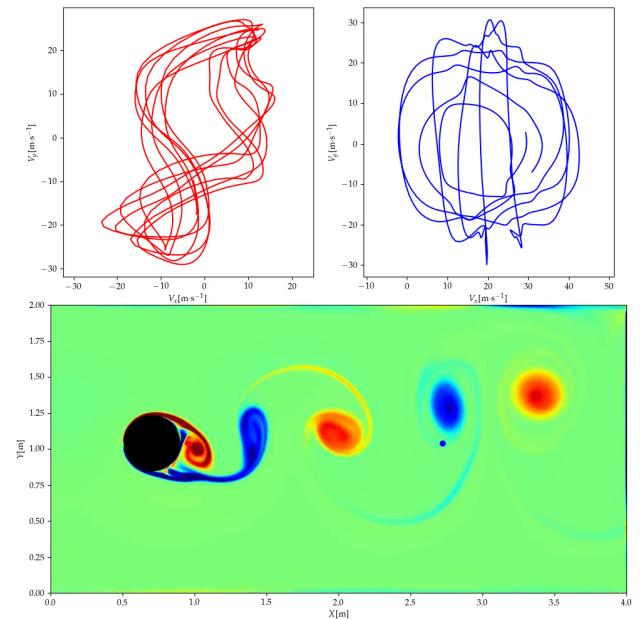


FIGURE 14. Rotationnel d'un écoulement chaotique $Re \approx 1000$

Les comportements chaotiques sont très riches et nous aurions aimé plus étudier ces simulation malheureusement ces simulations arrivent à la limite de ce qui est faisable techniquement avec le matériel à notre disposition.¹⁸

Conclusion

Dans le cadre de ce projet, nous avons pu implémenter la méthode de Boltzmann sur réseau, ce fut une aventure très riche et surtout très intéressante. Les résultats obtenus prouvent la polyvalence de la méthode pour simuler de nombreux systèmes. La méthode lattice Boltzmann ne bénéficie pas encore d'une documentation claire, ni standardisée, ni rigoureuse¹⁹ ce qui peut être très frustrant quand on cherche de la documentation sur le sujet. Il faut garder en tête que c'est un domaine relativement récent. Est il constitue un domaine de recherche très

18. les simulations des figures 14 et 13 on nécessite chacune plus de 120 Go de stockage et plus de 6h de simulations.

19. En tout cas si elle existe elle n'est pas disponible.

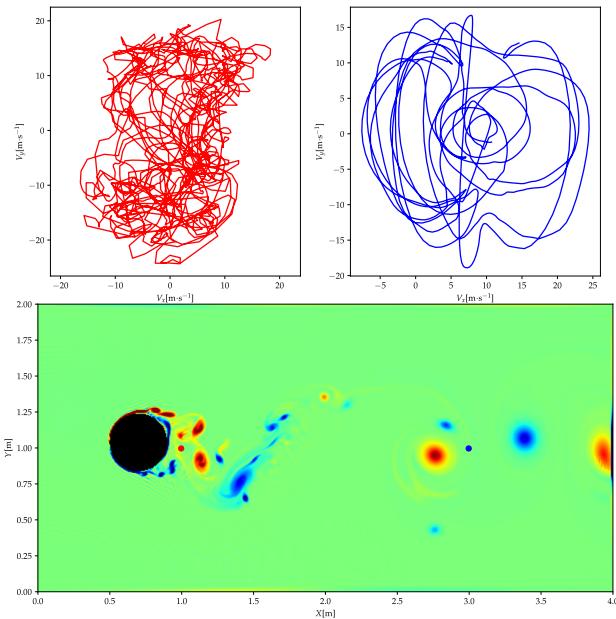


FIGURE 15. Rotationnel d'un écoulement chaotique $Re \approx 4000$

actif. Les LBM peuvent implémenter bien plus que seulement la physique des fluides et nous aurions aimé implémenter de la thermodynamique et des interactions entre deux fluides mais le temps et la contingence ne nous ont pas permis d'aller aussi loin que souhaité mais la fin de ce projet ne marque pas la fin de ce travail. Le GitHub associé à ce projet restera actif encore pour un moment, avec même un site web associé en cours de développement.

Références

- P. Martín and J. Puerta. *Plasma Physics*. Springer Netherlands, 1999.
- Wikipédia. Équations de Navier-Stokes — Wikipédia, l'encyclopédie libre, 2020. [En ligne; Page disponible le 5-décembre-2020].
- Wikipedia contributors. Millennium Prize Problems — Wikipedia, The Free Encyclopedia, 2020. [Online; accessed 9-December-2020].
- J.C.R. Hunt. LEWIS FRY RICHARDSON AND HIS CONTRIBUTIONS TO MATHEMATICS, METEOROLOGY, AND MODELS OF CONFLICT. *Annual Review of Fluid Mechanics*, 30(1) :xiii–xxxvi, January 1998.
- D. D'HUMIERES, Y. POMEAU, and P. LALLEMAND. Simulation d'allées de Von Karman bidimensionnelles à l'aide d'un gaz sur réseau. *Comptes rendus de l'Académie des sciences. Série 2, Mécanique, Physique, Chimie, Sciences de l'univers, Sciences de la Terre*, 1985.
- D. d'Humières, P. Lallemand, and U. Frisch. Lattice Gas Models for 3D Hydrodynamics. *Europhysics Letters (EPL)*, 2(4) :291–297, aug 1986.
- U. Frisch, B. Hasslacher, and Y. Pomeau. Lattice-Gas Automata for the Navier-Stokes Equation. *Phys. Rev. Lett.*, 56 :1505–1508, Apr 1986.
- Dominique d'Humières, Manfred Krafczyk, Li-Shi Luo, and Robert Rubinstein. Dedication to Pierre Lallemand on the occasion of his retirement. *Computers & Mathematics with Applications*, 58(5) :821 – 822, 2009. Mesoscopic Methods in Engineering and Science.
- S. Succi. *The Lattice Boltzmann Equation: For Fluid Dynamics and Beyond*. Numerical Mathematics and Scientific Computation. Clarendon Press, 2001.
- Marc B. Reider and James D. Sterling. Accuracy of discrete-velocity BGK models for the simulation of the incompressible Navier-Stokes equations. *Computers & Fluids*, 24(4) :459 – 467, 1995.
- Xiaoyi He and Li-Shi Luo. Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation. *Phys. Rev. E*, 56 :6811–6817, Dec 1997.
- Qisu Zou and Xiaoyi He. On pressure and velocity boundary conditions for the lattice Boltzmann BGK model. *Physics of Fluids*, 9(6) :1591–1598, Jun 1997.

Code FORTRAN

```

1 ! Author : Pacidus
2 ! Started at: 31.10.2020 00:17:57
3
4 !=====
5 !           Fluid Constants.
6 !=====
7 Module Fconst
8   Implicit None
9   Real(8), Parameter :: cs = 3.4d2      ! m·s-1 Speed of sound
10  Real(8), Parameter :: rho = 1.177d0    ! kg·m-3 Fluid density
11  Real(8), Parameter :: nu = .001       ! m2·s-1 Kinematic viscosity
12 End Module Fconst
13
14
15 !=====
16 !           Simulation Constants.
17 !=====
18 Module Sconst
19   Implicit None
20   Real(8), Parameter :: lx = 4d0        ! m     Length of the box
21   Real(8), Parameter :: ly = 2d0        ! m     Height of the box
22   Real(8), Parameter :: t = 5d0         ! s     Total time of the simulation
23   Real(8), Parameter :: sp = 10d0       ! m·s-1 Inlet Speed
24 End Module Sconst
25
26
27 !=====
28 !           Discretisation Constants.
29 !=====
30 Module Dconst
31   Use Fconst, Only: cs
32   Use Sconst, Only: lx, ly, t
33   Implicit None
34   Real(8), Parameter :: dt = 12d-6!5    ! s Timestep
35   Real(8), Parameter :: sqrt3 = Dsqrt(3d0) ! √3 No reason except aesthetic
36   Real(8), Parameter :: c = cs*sqrt3     ! m·s-1 Lattice speed
37   Real(8), Parameter :: dl = c*dt        ! m Spatial step
38   Integer(4), Parameter :: H = Int(ly/dl) ! Number of height-step
39   Integer(4), Parameter :: L = Int(lx/dl) ! Number of length-step
40   Integer(4), Parameter :: N = Int(t/dt)  ! Number of timestep
41 End Module Dconst
42
43
44 !=====
45 !           Computational Constants.
46 !=====
47 Module Cconst
48   Use Fconst, Only: cs, nu
49   Use Dconst, Only: dt, c
50   Use Sconst, Only: sp
51   Implicit None
52   Real(8), Parameter :: cs2 = cs*cs      ! m2·s-2 Sound speed squared
53   Real(8), Parameter :: ics2 = 1d0/cs2    ! s2·m-2 Inverse of cs2
54   Real(8), Parameter :: clean = nu*ics2/dt ! √ No reason except aesthetic
55   Real(8), Parameter :: tau = clean+5d-1  ! √ Characteristic timescale
56   Real(8), Parameter :: itau = 1d0/tau     ! √ Inverse of tau
57   Real(8), Parameter :: mitau = 1d0-itau   ! √ One minus inverse of tau
58   Real(8), Parameter :: zh1 = c/(c-sp)    ! √ Zou/He parameter
59   Real(8), Parameter :: zh2 = sp/(3d0*c)   ! √ another Zou/He parameter
60   Real(8), Parameter :: sp2 = sp*sp       ! m2·s-2 Outlet speed squared
61 End Module Cconst

```

```

62
63
64 !=====
65 !          Lattice Constants.
66 !=====
67 Module Lconst
68   Use Cconst, Only: c
69   Implicit None
70   Integer(1), Parameter :: D      = 2      ! D2 Number of spatial dimension
71   Integer(1), Parameter :: Q      = 9      ! Q9 Number of speed discretization
72   Integer(1), Parameter :: ed(Q,D) = &      ! # Directions of the speeds
73     reshape(&
74       (/0, 1, 0, -1, 0, 1, -1, -1, 1,&
75        0, 0, 1, 0, -1, 1, 1, -1, -1/),&
76       (/Q,D/))
77   Real(8), Parameter :: e(Q,D) = c*dble(ed) ! m·s-1 Speeds vectors
78   Real(8), Parameter :: w(Q)   = &           ! # D2Q9 Weights
79   (/4d0, 1d0, 1d0, 1d0, 1d0, 1d0/4d0,&
80    1d0/4d0, 1d0/4d0, 1d0/4d0/)/9d0
81 End Module Lconst
82
83 !=====
84 !          Simulation Variables.
85 !=====
86 Module Var
87   Implicit None
88   Integer(4), Allocatable :: Obj(:, :, :)      ! # coordinate of the Object
89   Real(8), Allocatable :: Feq(:, :, :, :)      ! kg·m-3 Density speed distribution
90   Real(8), Allocatable :: Usqr(:, :, :)        ! m·s-1 Macroscopic speed squared
91   Real(8), Allocatable :: F(:, :, :, :)        ! kg·m-3 Density speed distribution
92   Real(8), Allocatable :: U(:, :, :, :)        ! m·s-1 Macroscopic speed
93   Real(8), Allocatable :: Rho(:, :, :)         ! kg·m-3 Macroscopic density
94   Integer(4) :: s                            ! # Number of element in Obj
95 End Module Var
96
97
98 !=====
99 !          The main program.
100 !=====
101 Program LBM2D
102   Use Var
103   Use Dconst, Only: L, H, N
104   Implicit None
105   Integer(4) :: dN, d = 100, ni = 0
106   ! Real(4) :: time1, time2, timeTot
107   Call Allocatall
108   Call InitF
109
110  ! timeTot = 0
111  Do dN = 0, N
112    If (Modulo(dN, d) == 0) Then
113      Call Savefile(ni)
114      ni = ni + 1
115    End If
116
117  ! CALL CPU_TIME(time1)
118  CALL IOlet
119  CALL Bound
120  CALL Stream
121  CALL CMacro
122  CALL CFEq
123  CALL Collide
124  ! CALL CPU_TIME(time2)
125  ! timeTot = timeTot + (time2-time1)

```

```

126 ! Print *, real(H*L*(1+dN))/timetot
127 End Do
128
129 Call Deallocatall
130
131 ! Print *, real(H*L*N)/timetot
132 ! Print *, timetot
133 End Program LBM2D
134
135
136 !=====
137 ! Subroutines
138 !=====
139
140 !=====
141 ! Inlet & Outlet
142 ! Using Zou/He boundary condition to implement Dirichlet
143 !=====
144 Subroutine IOlet
145   Use Cconst, Only: zh1, zh2, sp
146   Use Var, Only: F, Rho, U
147   Use Dconst, Only: L, H, c
148   Use Sconst, Only: sp
149   Implicit None
150   Real(8) :: sca1, sca2, ux!, r
151   Integer(4) :: j!, i
152
153   ! Outflow condition
154   ! Top & Bottom wall
155   ! Do i = 1, L
156   !   F(7,i,1) = F(7,i,2)
157   !   F(3,i,1) = F(3,i,2)
158   !   F(6,i,1) = F(6,i,2)
159   !   F(8,i,H) = F(8,i,H-1)
160   !   F(5,i,H) = F(5,i,H-1)
161   !   F(9,i,H) = F(9,i,H-1)
162   ! End Do
163   ! Right wall
164   sca2 = 1d0/(3d0*c)
165   Do j = 1, H
166     sca1 = F(1,L,j)+F(3,L,j)+F(5,L,j)+ (2d0*(F(6,L,j) + F(2,L,j) + F(9,L,j)))
167     ux = c*(1d0 - sca1)
168
169     F(4,L,j) = F(2,L,j) + 2*ux*sca2
170     sca1 = F(5,L,j)-F(3,L,j)
171     F(7,L,j) = F(9,L,j) + (ux*sca2 + sca1)/2d0
172     F(8,L,j) = F(6,L,j) + (ux*sca2 - sca1 )/2d0
173   End Do
174
175   ! Inflow condition Left wall
176   Do j = 1, H
177     U(1,1,j) = sp
178     U(2,1,j) = 0d0
179
180     Rho(1,j) = 1
181
182     sca2 = zh2
183     F(2,1,j) = F(4,1,j) + sca2*2d0
184
185     sca1 = 1d0/2d0*(F(5,1,j) - F(3,1,j))
186     sca2 = sca2/2d0
187     F(6,1,j) = F(8,1,j) + sca2 + sca1
188     F(9,1,j) = F(7,1,j) + sca2 - sca1
189   End Do

```

```

190 End Subroutine
191
192
193 !=====
194 !          Boundary with collision
195 !=====
196 Subroutine Bound
197   Use Dconst, Only: L, H
198   Use Var, Only: F, Obj, s
199   Implicit None
200   Integer(4) :: n, i, j
201   Real(8) :: fi
202
203   Do n = 1, s
204     i = Obj(1,n)
205     j = Obj(2,n)
206
207     fi = F(2,i,j)
208     F(2,i,j) = F(4,i,j)
209     F(4,i,j) = fi
210
211     fi = F(6,i,j)
212     F(6,i,j) = F(8,i,j)
213     F(8,i,j) = fi
214
215     fi = F(3,i,j)
216     F(3,i,j) = F(5,i,j)
217     F(5,i,j) = fi
218
219     fi = F(7,i,j)
220     F(7,i,j) = F(9,i,j)
221     F(9,i,j) = fi
222   End Do
223
224 End Subroutine
225
226
227 !=====
228 !          Compute Macro
229 !=====
230 Subroutine CMacro
231   Use Var, Only: F, Rho, U, Usqr
232   Use Dconst, Only: L, H
233   Use Lconst, Only: Q, e
234   Implicit None
235   Integer(4) :: i, j
236   Integer(1) :: k
237   Real(8) :: r
238
239   Do j = 1, H
240     Do i = 1, L
241       Rho(i,j) = 0
242       U(1,i,j) = 0
243       U(2,i,j) = 0
244       Do k = 1, Q
245         Rho(i,j) = Rho(i,j) + F(k,i,j)
246         If(e(k,1) /= 0) Then
247           U(1,i,j) = U(1,i,j) + F(k,i,j)*e(k,1)
248         End If
249         If(e(k,2) /= 0) Then
250           U(2,i,j) = U(2,i,j) + F(k,i,j)*e(k,2)
251         End If
252       End Do
253       r = 1d0/Rho(i,j)

```

```

254     U(1,i,j) = U(1,i,j)*r
255     U(2,i,j) = U(2,i,j)*r
256     Usqr(i,j) = U(1,i,j) * U(1,i,j) + U(2,i,j) * U(2,i,j)
257   End Do
258 End Do
259 End Subroutine
260
261
262 ! =====
263 !           Compute Feq
264 ! =====
265 Subroutine CFeq
266   Use Var, Only: Feq, U, Usqr, Rho
267   Use Lconst, Only: Q, e, w
268   Use Dconst, Only: L, H
269   Use Cconst, Only: ics2
270   Implicit None
271   Real(8) :: Ue
272   Integer(4) :: i, j
273   Integer(1) :: k
274
275   Do j = 1, H
276     Do i = 1, L
277       Feq(i,j) = w(i)*Rho(i,j)*(1d0-.5d0*Usqr(i,j)*ics2)
278     End Do
279   End Do
280
281   Do j = 1, H
282     Do i = 1, L
283       Do k = 2, Q
284         Ue = e(k,1)*U(1,i,j)+e(k,2)*U(2,i,j)
285         Feq(k,i,j) = w(k)*Rho(i,j)*(1d0+(Ue+.5d0*((Ue*Ue*ics2)-Usqr(i,j)))*ics2)
286       End Do
287     End Do
288   End Do
289 End Subroutine
290
291
292 ! =====
293 !           Collide Step
294 ! =====
295 Subroutine Collide
296   Use Var, Only: F, Feq
297   Use Cconst, Only: itau, mitau
298   Use Dconst, Only: L, H
299   Use Lconst, Only: Q
300   Implicit None
301   Integer(4) :: i, j
302   Integer(1) :: k
303
304   Do j = 1, H
305     Do i = 1, L
306       Do k = 1, Q
307         F(k,i,j)= mitau*F(k,i,j) + itau*Feq(k,i,j)
308       End Do
309     End Do
310   End Do
311 End Subroutine
312
313
314 ! =====
315 !           Stream Step
316 ! =====
317 Subroutine Stream

```

```

318 Use Lconst, Only: Q, ed
319 Use Dconst, Only: L, H
320 Use Var, Only: F
321 Implicit None
322 Integer(1) :: k, e
323 Integer(4) :: i, j
324 Real(8) :: fs
325
326 Do k = 2, Q
327   If (ed(k,1) /= 0) Then
328     e = (ed(k,1)+1)*5d-1
329     Do j = 1, H
330       fs = F(k,1+e*(L-1),j)
331       Do i = 2+e*(L-3), L-e*(L-1), -ed(k,1)
332         F(k,i,j) = F(k,i-ed(k,1),j)
333       End Do
334     F(k,L-e*(L-1),j) = fs
335   End Do
336 End If
337 If (ed(k,2) /= 0) Then
338   e = (ed(k,2)+1)*5d-1
339   Do i = 1, L
340     fs = F(k,i,1+e*(H-1))
341     Do j = 2+e*(H-3), H-e*(H-1), -ed(k,2)
342       F(k,i,j) = F(k,i,j-ed(k,2))
343     End Do
344   F(k,i,H-e*(H-1)) = fs
345 End Do
346 End If
347 End Do
348
349 End Subroutine
350 !=====
351 !           Initialisation of F
352 !=====
353 Subroutine InitF
354   Use var, Only: U, Rho, F, Feq, Usqr
355   Use Cconst, Only: sp2
356   Use Sconst, Only: sp
357   Rho = 1
358   U(1,:,:)=sp
359   U(2,:,:)=0
360   Usqr = sp2
361   Call CFeq
362   F = Feq
363 End Subroutine
364
365
366 !=====
367 !           Convert the image of the object from svg to pgm
368 !           and import it into a matrix with 1 and 0 (1 stand for the object)
369 !=====
370 Subroutine Object
371   Use Dconst, Only: L, H
372   Use Var, Only: Obj, s
373   Implicit None
374   Character(len=210) :: command          ! The bash command
375   Integer(1) :: Ob(L,H)
376   Integer(4) :: i, j                      ! Itterator
377
378   command = ('("rsvg-convert -w ",i0," -h ",i0," objet.svg -o objet.png")'
379   Write(command, command) L, H
380
381   Call execute_command_line(command, wait=.true.)

```

```

382
383 command = "convert -compress none objet.png -alpha extract -threshold 0%&
384 & -negate objet.pbm"
385 Call execute_command_line(command, wait=.true.)
386
387 Open(10, file = 'objet.pbm', action='read')
388
389 Read(10, *) command
390 Read(10, *) command
391
392 Do i=1, H
393   Read(10, *) Ob(:, i)
394 End Do
395 Close(10)
396
397 s = 0
398 Do j=1, H
399   Do i=1, l
400     s = s + Ob(i, j)
401   End Do
402 End Do
403
404 Allocate(Obj(2,s))
405
406 s = 0
407 Do j=1, H
408   Do i=1, l
409     If (Ob(i, j) == 1) Then
410       s = s + 1
411       Obj(1,s) = i
412       Obj(2,s) = j
413     End If
414   End Do
415 End Do
416
417 End Subroutine
418
419
420 !=====
421 !           Save the macroscopic values such as speed and density
422 !=====
423 Subroutine Savefile(dN)
424   Use Var, Only: U, Rho
425   Use Dconst, Only: H
426   Implicit None
427   Character(len=24) :: Namefile
428   Integer(4) :: dN, i
429
430   Write(Namefile, '("./Ux/",i5.5,".csv")') dN
431   Open(10, file=Namefile, action='Write')
432   Write(Namefile, '("./Uy/",i5.5,".csv")') dN
433   Open(11, file=Namefile, action='Write')
434   Write(Namefile, '("./P/",i5.5,".csv")') dN
435   Open(12, file=Namefile, action='Write')
436   Do i=1, H
437     Write(10, *) U(1,:,i)
438     Write(11, *) U(2,:,i)
439     Write(12, *) Rho(:,i)
440   End Do
441   Close(10)
442   Close(11)
443   Close(12)
444 End Subroutine
445

```

```
446
447
448 !=====
449 !          Allocate all matrixes
450 !=====
451 Subroutine Allocatall
452   Use Var, Only: F, Feq, U, Usqr, Rho
453   Use Dconst, Only: L, H
454   Use Lconst, Only: Q, D
455
456   Allocate(F(Q,L,H))
457   Allocate(Feq(Q,L,H))
458   Allocate(U(D,L,H))
459   Allocate(Usqr(L,H))
460   Allocate(Rho(L,H))
461   Call Object
462 End Subroutine
463
464
465 !=====
466 !          Deallocate all matrixes
467 !=====
468 Subroutine Deallocatall
469   Use Var
470
471   Deallocate(F)
472   Deallocate(Feq)
473   Deallocate(U)
474   Deallocate(Usqr)
475   Deallocate(Rho)
476   Deallocate(Obj)
477 End Subroutine
```

Code PYTHON pour les animations et l'analyse des données

Animation fluide

```

1  #!/usr/bin/env python3
2  # coding:utf-8
3
4  import os
5  import subprocess
6  import numpy as np
7  import matplotlib.pyplot as plt
8  from matplotlib import rc
9
10 rc("font",**{"family":"serif","serif":["Palatino"]})
11 rc('text', usetex=True)
12
13 PUx = './Ux/'
14 PUy = './Uy/'
15 PP = './P/'
16
17 files = sorted(os.listdir(PP))
18 imgs = sorted(os.listdir("./anim/"))
19 for f in files:
20     if f.split(".")[0]+".png" not in imgs:
21         Ux = np.loadtxt(PUx+f)
22         dUx = np.gradient(Ux, axis=0)
23         Uy = np.loadtxt(PUy+f)
24         dUy = np.gradient(Uy, axis=1)
25         #     plt.imsave('./anim/'+f.split(".")[0]+".png",np.sqrt(Uy*Uy+Ux*Ux), vmin=0, cmap='jet')
26         plt.imsave('./anim/'+f.split(".")[0]+".png", (dUy-dUx),vmin = -5e0, vmax=5e0, cmap='jet')
27     a = np.loadtxt(PP+f)
28     if np.isnan(a).any():
29         #
30         break
31     #     plt.imshow(a, cmap='jet')
32     #     plt.pause(0.1)
33     #     plt.cla()
34     #     plt.clf()
#    print(np.loadtxt(PUx+f).mean())

```

Animation analyse Poiseuille

```

1 import numpy as np
2 import cv2
3 import matplotlib.pyplot as plt
4
5 plt.rc("font",**{"family":"serif","serif":["Palatino"]})
6 plt.rc('text', usetex=True)
7
8 figure = plt.figure(figsize=(10, 10), tight_layout=True)
9
10 ax1 = figure.add_axes([0.06, 0.025, 0.92, 0.5])
11 ax2 = figure.add_axes([0.06, 0.55, 0.3, 0.43])
12 ax3 = figure.add_axes([0.43, 0.55, 0.55, 0.215])
13 ax4 = figure.add_axes([0.43, 0.765, 0.55, 0.215], sharex=ax3)
14
15 ax1.set_xlabel(r"$X[m]$")
16 ax1.set_ylabel(r"$Y[m]$")
17 ax2.set_xlabel(r"$V_x[m\cdot s^{-1}]$")
18 ax2.set_ylabel(r"$Y[m]$")
19 ax3.set_xlabel(r"$t[s]$")
20 ax3.set_ylabel(r"$V[m^3/s]$")
21 ax4.set_ylabel(r"$\Delta V[m^3/s]$")
22 #ax3.set_xscale("log")
23 #ax3.set_yscale("symlog")
24 #ax4.set_yscale("symlog")
25 plt.setp(ax4.get_xticklabels(), visible=False)
26
27 cap = cv2.VideoCapture('rot.mp4')
28
29 fr = 0
30 namedata = ["./Ux/%05d.csv", "./Uy/%05d.csv"]
31
32 x1 = []
33 y1 = []
34 x2 = []
35 y2 = []
36
37 Poiseuille = lambda Dv, h: Dv*6/h
38 speedth = lambda Vmax, y, h: Vmax*(y-(y*y)/h)/h
39
40 ret, frame = cap.read()
41 rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
42
43 L = rgb.shape
44 l = [2/L[0],4/L[1]]
45 xcoord = [int(.75/l[1]),int(2.5/l[1])]
46
47 def constr():
48     indices = rgb[:,j].sum(1).nonzero()[0]
49     d = indices[0]-1
50     f = indices[-1]+1
51     return(np.array([d]+list(indices)+[f]))
52
53 j = xcoord[0]
54 y1coord = constr()
55 j = xcoord[1]
56 y2coord = constr()
57
58 y1 = l[0]*y1coord
59 y2 = l[0]*y2coord
60
61 extent = [0, 4, 0, 2]

```

```

62 img = ax1.imshow(rgb, extent=extent)
63
64 yth = y1
65 h1 = yth[-1] - yth[0]
66
67 yth = y2
68 h2 = yth[-1] - yth[0]
69 def load(fr, rgb):
70     U = np.loadtxt(namedata[0] % fr)
71     V = np.loadtxt(namedata[1] % fr)
72     Vx1 = U[y1coord, xcoord[0]]
73     Vx2 = U[y2coord, xcoord[1]]
74     Vx1[[0, -1]] = 0
75     Vx2[[0, -1]] = 0
76     return(Vx1[::-1], Vx2[::-1])
77
78 Vx1, Vx2 = load(fr,rgb)
79
80 line1 = ax2.plot(Vx1,y1, 'r')[0]
81 line2 = ax2.plot(Vx2,y2, 'b')[0]
82
83 ax1.plot(l[0]*xcoord[0]+y1*0,y1, 'r-')
84 ax1.plot(l[0]*xcoord[1]+y2*0,y2, 'b-')
85
86 dt = 5e-3
87 V1 = [Vx1.sum()*l[0]]
88 V2 = [Vx2.sum()*l[0]]
89 DV = [V2[-1]-V1[-1]]
90 t = [fr*dt]
91 line3 = ax3.plot(V1,t, 'r')[0]
92 line4 = ax3.plot(V2,t, 'b')[0]
93
94 line5 = ax4.plot(DV,t, '#aa00aa')[0]
95
96 vmax = Poiseuille(V1[-1],h2)
97 Vxth = speedth(vmax, yth-yth[0], h2)
98 line6 = ax2.plot(Vxth, yth, 'k--', label="Écoulement théorique")[0]
99 ax2.legend()
100 while(cap.isOpened()):
101
102     plt.savefig("./anim2/%05d.svg"%fr)
103
104     fr += 1
105
106     ret, frame = cap.read()
107     rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
108
109     Vx1, Vx2 = load(fr, rgb)
110     line1.set_xdata(Vx1)
111     line2.set_xdata(Vx2)
112     vmax = Poiseuille(V1[-1],h2)
113     Vxth = speedth(vmax, yth-yth[0], h2)
114     line6.set_xdata(Vxth)
115     V1.append(Vx1.sum()*l[0])
116     V2.append(Vx2.sum()*l[0])
117     t.append(fr*dt)
118     line3.set_ydata(V1)
119     line3.set_xdata(t)
120     line4.set_ydata(V2)
121     line4.set_xdata(t)
122
123     DV.append(V2[-1]-V1[-1])
124     line5.set_ydata(DV)
125     line5.set_xdata(t)

```

```
126
127     ax2.relim()
128     ax2.autoscale_view()
129     ax3.relim()
130     ax3.autoscale_view()
131     ax4.relim()
132     ax4.autoscale_view()
133
134     img.set_array(rgb)
135
136 plt.savefig("./anim2/%05d.svg"%fr)
137
138 img.set_array(rgb)
139
140 cap.release()
141 cv2.destroyAllWindows()
```

Animation analyse Von Karmann

```

1 import numpy as np
2 import cv2
3 import matplotlib.pyplot as plt
4
5 plt.rc("font",**{"family":"serif","serif":["Palatino"]})
6 plt.rc('text', usetex=True)
7
8 figure = plt.figure(figsize=(10, 10), tight_layout=True)
9
10 ax1 = figure.add_axes([0.06, 0.025, 0.92, 0.5])
11 ax2 = figure.add_axes([0.08, 0.55, 0.4, 0.43])
12 ax3 = figure.add_axes([0.56, 0.55, 0.4, 0.43])
13
14 ax1.set_xlabel(r"$X[m]$")
15 ax1.set_ylabel(r"$Y[m]$")
16 ax2.set_xlabel(r"$V_x[m\cdot s^{-1}]$")
17 ax2.set_ylabel(r"$V_y[m\cdot s^{-1}]$")
18 ax3.set_xlabel(r"$V_x[m\cdot s^{-1}]$")
19 ax3.set_ylabel(r"$V_y[m\cdot s^{-1}]$")
20
21 ax2.axis('equal')
22 ax3.axis('equal')
23
24 cap = cv2.VideoCapture('rot.mp4')
25
26 fr = 0
27 namedata = ["./Ux/%05d.csv", "./Uy/%05d.csv"]
28
29 x1 = []
30 y1 = []
31 x2 = []
32 y2 = []
33
34 def load(x1,y1,x2,y2,fr):
35     U = np.loadtxt(namedata[0]%fr)
36     V = np.loadtxt(namedata[1]%fr)
37     x1.append(U[coord[0]])
38     y1.append(V[coord[0]])
39     x2.append(U[coord[1]])
40     y2.append(V[coord[1]])
41
42 ret, frame = cap.read()
43 rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
44
45 L = rgb.shape
46 l = [2/L[0],4/L[1]]
47 coord = [(int(1/l[0]),int(1/l[1])),(int(1/l[0]),int(3/l[1]))]
48 extent = [0, 4, 0, 2]
49 img = ax1.imshow(rgb, extent=extent)
50 load(x1,y1,x2,y2,fr)
51
52 line1 = ax2.plot(x1,y1,'r')[0]
53 line2 = ax3.plot(x2,y2,'b')[0]
54
55 ax1.plot([l[1]*coord[0][1]],[l[0]*coord[0][0]], 'ro')
56 ax1.plot([l[1]*coord[1][1]],[l[0]*coord[1][0]], 'bo')
57
58 while(cap.isOpened()):
59
60     plt.savefig("./anim2/%05d.svg"%fr)
61

```

```
62     fr += 1
63
64     load(x1,y1,x2,y2,fr)
65     line1.set_xdata(x1)
66     line1.set_ydata(y1)
67     line2.set_xdata(x2)
68     line2.set_ydata(y2)
69     ax2.relim()
70     ax2.autoscale_view()
71     ax3.relim()
72     ax3.autoscale_view()
73     img.set_array(rgb)
74
75     if (fr%2 == 0) or (fr%7 == 0):
76         x1 = x1[1:]
77         y1 = y1[1:]
78         x2 = x2[1:]
79         y2 = y2[1:]
80
81     ret, frame = cap.read()
82     rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
83
84     if cv2.waitKey(1) & 0xFF == ord('q'):
85         break
86
87 plt.savefig("./anim2/%05d.pdf"%fr)
88
89 img.set_array(rgb)
90
91 cap.release()
92 cv2.destroyAllWindows()
```

Automatisation et animations

```
1 MAKEFLAGS += --no-print-directory
2
3 make: windtunnel.f90
4     gfortran windtunnel.f90 -O3 -Wall -o prog
5     make clean
6
7 Rmake: Rewindtunnel.f90
8     gfortran Rewindtunnel.f90 -O3 -Wall -o prog
9     make clean
10
11 lmake: windtunnel.f90
12     gfortran windtunnel.f90 -O3 -mcmodel=large -o prog
13     make clean
14
15 movie: ./anim/*.png
16     @echo y | ffmpeg -r 25 -i "./anim/%05d.png" -loop 1 -i "./objet.pbm" \
17         -filter_complex "color=white:s=10x10[th];\
18             [th][0]scale2ref=w=iw:h=ih[th][v];[1][th][1][v]threshold" -qscale:v 0 \
19             -c:v libvpx-vp9 -crf 0 rot.mp4
20
21 out: ./anim2/*.svg
22     @echo y | ffmpeg -r 30 -i "./anim2/%05d.svg" -qscale:v 0 -c:v libvpx-vp9 \
23         -vf scale=1080:1080 -crf 20 out.mp4
24
25 clean:
26     rm *.mod
27
28 cleandta:
29     rm ./P/*.csv
30     rm ./Ux/*.csv
31     rm ./Uy/*.csv
```