# Comparaison entre méthodes de simulation de fluides

Michel Jean Joseph Donnet

Invalid Date

# Table des matières

# Chapitre 1

# Introduction

La terre est appelée “La planète bleue”. En effet, près de 70% de la surface de la terre est recouverte d’eau, donnant ainsi une couleur bleue à la terre, comme ont pu constater les astronautes lors de la mission Apollo en 1972. De plus, la terre possède l’eau dans tous ses états: solide, liquide et gazeux.

L’eau est quelque chose qui a beaucoup intéressé les scientifiques de toutes les époques. Nous pouvons notamment citer parmi eux le légendaire Archimède qui, au cours du 3ème siècle avant Jésus-Christ, utilisa le principe de la poussée d’Archimède afin de déterminer si une couronne était en or selon la légende.

Au 17ème siècle, un mathématicien et physicien dénommé Galilée s’est penché sur les propriétés des fluides et a découvert que la viscosité est une propriété fondamentale d’un fluide.

Puis entre le 17ème et le 18ème siècle s’est démarqué le célèbre Isaac Newton en développant plus les travaux de Galilée et en donnant la définition d’un fluide newtonien, qui est un fluide dont la viscosité est inépendante de la contrainte mécanique s’exerçant sur celui-ci, donc un fluide ayant un comportement prévisible.

Grâce aux études d’Isaac Newton, le renommé Leonhard Euler d’origine bâloise établit au cours du 18ème siècle des équations 2.2 modélisant l’écoulement d’un fluide parfait adiabatique, donc un fluide où la viscosité et les effets de la chaleur ne sont pas pris en compte.

Mais cela ne satisfit pas tous les scientifiques. C’est pourquoi, au cours du 19ème siècle, frustrés par l’impossibilité de modéliser des fluides visqueux, le mathématicien Henri Navier et le physicien Georges Gabriel Stokes décidèrent d’ajouter la notion de viscosité aux équations d’Euler 2.2, étendant ainsi les équations sur les fluides newtonien. Leur travail fut reconnu et utilisé sous le nom d’équations de Navier-Stokes 2.3. Même de nos jours, personne n’a réussi à trouver une forme analytique aux équations de Navier-Stokes 2.3, c’est pourquoi elles font toujours partie des 7 problèmes du prix du millénaire.

Cependant, la technologie a fait de nombreux progrès surtout vers la fin du 20ème siècle avec l’apparition de l’ordinateur, ce qui permit aux scientifiques de tenter de résoudre les équations de Navier-Stokes 2.3 grâce à des approximations et des méthodes numériques. Plusieurs méthodes ont donc été créées, notamment la méthode Smoothed Particle Hydrodynamics (SPH), la méthode Fluid-Implicit Particles (FLIP) et la méthode Lattice Boltzmann Method (LBM). Cependant, quelles sont les différences entre ces méthodes ? Y-a-t-il une méthode plus rapide qu’une autre ? Serait-il possible d’utiliser ces méthodes afin de faire du rendu en temps réel de haute qualité ?

Nous allons commencer par expliquer la théorie de chacune de ces méthodes, puis nous allons nous pencher sur les différences les distinguant et sur les résultats concrêt obtenu grâce au code de GVDB-Voxel et de FluidX3D.

# partie I

# Méthodologie

# Chapitre 2

# Mécanique des fluides

Un fluide est composé de nombreuses particules, qui peuvent autant être des atomes que des ions ou encore des molécules, qui sont des atomes liés entre eux par des liaisons fortes ou covalentes. À la différence d'un solide, un fluide est complètement déformable.

Il existe différent types de fluides, notamment:

- les gaz: ce sont des fluides composés de particules isolées mouvant en toute liberté et pouvant entrer en collision.
- les liquides: ce sont des fluides composés de particules liées entre elles par des liaisons faibles, comme les liaisons hydrogène. Les particules ne peuvent donc pas se mouvoir en toute liberté, et lorsqu'une particule bouge, elle exerce une influence sur les autres particules liées à elle.

## 2.1 Variables

Table 2.1: table des variables utilisées

| Variable | Description | Unité |
|---|---|---|
| $t$ | temps | $s$ |
| $\rho$ | Masse volumique du fluide | $kg \cdot m^{-3}$ |
| $\mathbf{v}$ | Vecteur vitesse du fluide | $m \cdot s^{-1}$ |
| $p$ | Pression du fluide | $N \cdot m^{-2}$ |
| $F$ | Forces externes s'appliquant sur le fluide | $N$ |
| $E$ | Énergie totale par unité de masse | $J \cdot kg^{-1}$ |
| $e$ | Énergie interne par unité de masse | $J \cdot kg^{-1}$ |
| $\Sigma$ | Contrainte de viscosité du fluide | $N \cdot m^{-2}$ |
| $q$ | Flux de chaleur causé par conduction thermique | $J \cdot s^{-1} \cdot m^{-2}$ |
| $q_R$ | Flux de chaleur causé par rayonnement | $J \cdot s^{-1} \cdot m^{-2}$ |
| $m$ | Masse des particules | $kg$ |
| $f$ | Fonction de densité de probabilité | $m^{-3} \cdot s^{-1}$ |
| $\mathbf{x}$ | Position de la particule | $m$ |

Dans la table 2.1, $E = e + \frac{1}{2}\|\mathbf{v}\|^2$

## 2.2 Équations d'Euler

Les équations d'Euler sont un ensemble d'équations décrivant l'écoulement d'un fluide non visqueux. Il y a 3 équations d'Euler.

### 2.2.1 Équation de continuité

L'équation de continuité peut être définie comme dans l'équation 2.1.

$$\frac{\partial}{\partial t}\rho + \nabla(\rho \mathbf{v}) = 0 \tag{2.1}$$

avec:

- $\frac{\partial}{\partial t}\rho$ nous donne la variation de la masse par unité de volume en fonction du temps
- $\nabla(\rho \mathbf{v})$ est le flux de masse. Il nous indique comment la masse se déplace et se redistribue dans le volume

Dans l'équation 2.1, nous pouvons voir que la variation de la masse doit être égale au flux de masse du fluide. Cela signifie que la masse du fluide suit le principe de conservation de la matière

### 2.2.2 Équation de la quantité de mouvement

L'équation de la quantité de mouvement est définie dans l'équation 2.2.

$$\frac{\partial}{\partial t}(\rho \mathbf{v}) + \nabla \cdot (\rho \mathbf{v}\mathbf{v}^T) = -\nabla p + \rho \cdot F \tag{2.2}$$

avec:

- $\rho \cdot F$ nous donne l'ensemble des forces externes s'appliquant par unité de volume
- $\nabla \cdot (\rho \mathbf{v}\mathbf{v}^T)$ représente le changement de vitesse dû au mouvement du fluide
- $\frac{\partial}{\partial t}(\rho \mathbf{v})$ est la variation temporelle de la quantité de mouvement

Dans l'équation 2.2, il y a d'une part la variation temporelle de la quantité de mouvement plus la répartition de la quantité de mouvement dans le fluide, et de l'autre part la force résultant des variations de la pression plus les autres forces externes. On peut donc reconnaître la 2ème loi de Newton disant que la quantité de mouvement est égal à la somme des forces.

### 2.2.3 Équation de l'énergie

Soit un liquide adiabatique, c'est à dire pour lequel la chaleur n'est pas prise en compte.

L'équation de l'énergie est alors donnée par l'équation 2.3.

$$\frac{\partial}{\partial t}\rho E + \nabla \cdot (\rho E \mathbf{v}) = -\nabla \cdot (p\mathbf{v}) + \rho F \mathbf{v} \tag{2.3}$$

avec:

- $\frac{\partial}{\partial t}\rho E$ est la variation temporelle de l'énergie par unité de volume.
- $\nabla \cdot (\rho E \mathbf{v})$ nous donne le flux d'énergie à travers le fluide, donc comment l'énergie est transportée dans le fluide
- $-\nabla \cdot (p\mathbf{v})$ est le travail de la pression sur le fluide
- $\rho F \mathbf{v}$ est le travail des forces extérieures sur le fluide

Dans l'équation 2.3, il y a d'une part la somme entre la variation et le flux d'énergie, et d'autre part la somme du travail des forces s'exerçant sur le fluide. Cela découle du principe de conservation d'énergie: la somme du travail des forces est égale à l'énergie du fluide.

## 2.3 Équations de Navier-Stokes

Les équations de Navier-Stokes sont basées sur les équations d'Euler 2.2. Elles y ajoutent la notion de viscosité, qui représente les forces de friction interne au fluide. Elles permettent donc de modéliser des fluides réels visqueux à la différence des équations d'Euler 2.2 qui modélisent des fluides parfaits.

Il y a également 3 équations de Navier-Stokes.

### 2.3.1 Équation de continuité

Celle-ci ne diffère pas de l'équation de continuité d'Euler 2.1. Ceci semble normal, car la masse du fluide, même dans un fluide visqueux, ne peut toujours pas être créée ni détruite, et suit toujours le principe de conservation de la matière.

### 2.3.2 Équation de la quantité de mouvement

L'équation de la quantité de mouvement de Navier-Stokes 2.4 est basée sur l'équation d'Euler 2.2.

$$\frac{\partial}{\partial t}(\rho \mathbf{v}) + \nabla \cdot (\rho \mathbf{v}\mathbf{v}^T) = -\nabla p + \nabla \Sigma + \rho \cdot F \tag{2.4}$$

- $\nabla \Sigma$ est la force exerçée par la viscosité du fluide

Dans l'équation 2.4, la force exercée par la viscosité a été ajoutée à la somme des forces de l'équation 2.2 d'Euler. Ainsi, l'équation 2.4 suit toujours la 2ème loi de Newton.

### 2.3.3 Équation de l'énergie

L'équation de l'énergie de Navier-Stokes 2.5 est basée sur l'équation de l'énergie d'Euler 2.3.

$$\frac{\partial}{\partial t}\rho E + \nabla \cdot (\rho E \mathbf{v}) = -\nabla \cdot (p\mathbf{v}) + \nabla \cdot \Sigma \mathbf{v} + \rho F \mathbf{v} + \nabla \cdot q + \nabla \cdot q_R \tag{2.5}$$

- $\nabla \cdot \Sigma \mathbf{v}$ est le travail de la viscosité du fluide
- $\nabla \cdot q + \nabla \cdot q_R$ est le travail de la chaleur sur le fluide

L'équation de l'énergie de Navier-Stokes 2.5 ajoute à l'équation d'Euler 2.3 le travail de la viscosité et de la chaleur sur le fluide. Ainsi, les fluides non adiabatiques sont également pris en compte par cette équation.

Dans la plupart des cas, l'équation de l'énergie n'est pas prise en compte lors de la simulation de fluides notamment à cause de la complexité du calcul.

## 2.4 Équation de Boltzmann

Les équations de Boltzmann sont utiles pour décrire le comportement de systèmes composés de particules, tel que les gaz ou les liquides.

Dans les équations de Boltzmann, chaque particule composant le système possède une vitesse et une position qui varient dans le temps en fonction des mouvements de la particule.

La fonction $f(\mathbf{x}, \mathbf{v}, t)$ donne alors la densité de probabilité de trouver une particule à une position $\mathbf{x}$ avec une vitesse $\mathbf{v}$ à un temps $t$.

L'objectif des équations de Boltzmann est donc de déterminer comment évolue cette fonction de densité de probabilité $f$ au cours du temps.

La fonction de distribution évolue suivant les forces extérieures s'appliquant sur les particules, les collisions entre les particules et la diffusion des particules, qui tend à avoir la même concentration de particules dans tout le système. C'est pourquoi, la variation de la fonction de distribution peut s'écrire comme dans l'équation 2.6.

$$\frac{\partial f}{\partial t} = \left(\frac{\partial f}{\partial t}\right)_{force} + \left(\frac{\partial f}{\partial t}\right)_{diff} + \left(\frac{\partial f}{\partial t}\right)_{coll} \tag{2.6}$$

### 2.4.1 Collisions négligées

Supposons qu'il n'y a pas de collisions entre les particules. Cela signifie que les particules n'interagissent pas entre elles, ce qui entraîne aucune modification de la fonction de distribution $f$. Ainsi, la dérivée totale de $f$ par rapport au temps $t$ est nulle: $\frac{df}{dt} = 0$ car $f$ ne varie pas dans le temps.

Grâce aux règles de chaînage, il est possible de trouver la dérivée totale de $f$ par rapport au temps $t$, comme montrée dans l'équation 2.7.

$$\begin{aligned}\frac{df}{dt} &= (\frac{dt}{dt}\cdot\frac{\partial}{\partial t} + \frac{d\mathbf{x}}{dt}\cdot\frac{\partial}{\partial \mathbf{x}} + \frac{d\mathbf{v}}{dt}\cdot\frac{\partial}{\partial \mathbf{v}})f(\mathbf{x},\mathbf{v},t) \\ &= (\frac{\partial}{\partial t} + \mathbf{v}\cdot\frac{\partial}{\partial \mathbf{x}} + \frac{F}{m}\cdot\frac{\partial}{\partial \mathbf{v}})f(\mathbf{x},\mathbf{v},t) \\ &= \frac{\partial f}{\partial t} + \mathbf{v}\nabla_{\mathbf{x}}f + \frac{F}{m}\nabla_{\mathbf{v}}f\end{aligned} \tag{2.7}$$

Dans l'équation 2.7, le terme $\frac{\partial f}{\partial t}$ montre le changement de $f$ à vitesse et position constante, le terme $\mathbf{v}\nabla_{\mathbf{x}}f$ représente le changement de $f$ dû aux déplacements des particules, et le terme $\frac{F}{m}\nabla_{\mathbf{v}}f$ représente le changement de $f$ dû aux forces extérieures, tel que la gravité, s'appliquant sur les particules. Ainsi, les déplacements des particules sont uniquement dûe aux forces extérieures et à la diffusion des particules, ce qui donne en combinant l'équation 2.7 avec le fait que $f$ ne varie pas en fonction du temps l'équation 2.8.

$$\frac{\partial f}{\partial t} + \mathbf{v}\nabla_{\mathbf{x}}f + \frac{F}{m}\nabla_{\mathbf{v}}f = 0 \tag{2.8}$$

### 2.4.2 Collisions non négligées

Cependant, dans la réalité, des collisions existent entre les particules. Comme les particules interagissent entre elles, la fonction de distribution va subir des modifications en fonction des collisions. C'est pourquoi la dérivée totale de $f$ par rapport à $t$ ne sera plus égal à 0, mais au terme $\left(\frac{\partial f}{\partial t}\right)_{coll}$ qui capture l'effet des interactions entre les particules, ce qui donne la modification de l'équation 2.8 en l'équation 2.9.

$$\frac{\partial f}{\partial t} + \mathbf{v}\nabla_{\mathbf{x}}f + \frac{F}{m}\nabla_{\mathbf{v}}f = \left(\frac{\partial f}{\partial t}\right)_{coll} \tag{2.9}$$

L'équation 2.9 est appelée équation de Boltzmann. Il ne reste plus qu'à définir le terme de collision $\left(\frac{\partial f}{\partial t}\right)_{coll}$, ce qui est une chose complexe qui ne va pas être traîtée ici.

# Chapitre 3

# Méthode SPH

## 3.1 Histoire

La méthode Smoothed Particle Hydrodynamics (SPH) a été inventée en 1977 par Bob Gingold et Joe Monaghan Gingold et Monaghan [8] et indépendamment par Leon Lucy Lucy [18] afin de simuler des phénomènes astrophysiques, tel que la formation et l'évolution d'une étoile ou d'une galaxie. Il s'agissait tout d'abord d'une approche probabiliste. Les équations de la mécanique des fluides pouvaient effectivement servir à décrire ce genre de phénomènes astrophysiques car il s'agit de gaz ou d'une multitude de corps évoluant d'une manière similaire à un liquide ou un gaz.

La méthode SPH s'est ensuite développée dans le domaine de la mécanique des fluides ou elle a servit notamment à modéliser non seulement des fluides compressibles et incompressibles, mais également des phénomènes thermiques et magnétiques.

Puis vers 1990, la méthode SPH a été étendue à la mécanique des structures afin de simuler par exemple des impact à forte vitesse ou des déchirures de matériaux grâce notamment au travail de Libersky et Petschek (citation ici).

De nos jour, la méthode SPH est encore utilisée dans la mécanique des fluides, mais également pour simuler des impacts haute vitesse, des fragmentations ou encore des explosions, si bien que le terme Hydrodynamics n'est plus adapté. Cependant, pour des raisons historiques, on conserve le terme Hydrodynamics.

## 3.2 Interpolation

La méthode SPH utilise une technique d'interpolation afin de déterminer le résultat des équations de Navier-Stokes 2.3.

L'interpolation utilisée dans la méthode SPH est basée sur le principe que la distribution de Dirac $\delta(r)$, appelée par abus de langage fonction Dirac et définie dans l'équation 3.2, peut être considérée comme l'élément neutre de la convolution, comme nous montre l'équation 3.1. Étant une distribution de probabilité, la fonction Dirac respecte la propriété d'une distribution, donc on a la propriété 3.3.

$$\begin{aligned} f * \delta(x) &= \int_y f(y)\delta(y-x) \\ &= f(x) \end{aligned} \tag{3.1}$$

avec:

$$\delta(x) = \begin{cases} 1 \text{ si } x = 0 \\ 0 \text{ sinon} \end{cases} \tag{3.2}$$

et:

$$\int \delta(x)dx = 1 \tag{3.3}$$

On définit donc une fonction noyau $W$ qui est une approximation de la fonction Dirac et qui respecte les propriétés 3.4 et 3.5.

$$\int_r W(r,h)dr = 1 \tag{3.4}$$

$$\lim_{h\to 0} W(r,h) = \delta(r) \tag{3.5}$$

L'équation 3.1 devient alors l'équation 3.6.

$$\begin{aligned} f * W(x) &= \int_y f(y)W(y-x) \\ &\approx f(x) \end{aligned} \tag{3.6}$$

## 3.3 Principes de base

La méthode SPH représente le fluide comme un ensemble de particules interagissant entre elles. Elle simule le comportement de chaque particule, donc il s'agit d'une méthode lagrangienne.

De plus, la méthode SPH est une méthode sans maillage, ce qui signifie en d'autres termes qu'elle ne nécessite pas de maillage fixe. Cela implique que la méthode SPH peut être utilisée avec une taille de domaine adaptative, et est donc particulièrement adaptée pour des problèmes complexes car on ne fera que la quantité de calcul nécessaire. Par exemple, si on pense à simuler un verre d'eau se renversant sur une table, la méthode SPH sera particulièrement adaptée car elle ne calculera que les endroits où le fluide est présent et pas toute la table comme le ferait une méthode avec maillage fixe qui devrait définir un domaine de calcul fixe et s'y tenir.

Elle ne tente pas de résoudre les équations du fluide dans une grille fixe…. (reformuler ?)

Comme la méthode SPH est une méthode sans maillage, elle utilise une technique d'interpolation afin de déterminer le résultat des équations aux dérivées partielles.

(à nettoyer !!=

La méthode SPH simule le fluide comme un ensemble de particules. Ces particules possèdent leurs caractéristiques propres tel que leur masse, leur position, leur vitesse. Une particule interagit avec les particules l'avoisinant à travers un noyau de lissage $W$. Cependant, une particule ne peut pas interagir avec toutes les particules… Par exemple, on ne veut pas qu'une particule à une extrémité du fluide interagisse avec une particule à l'autre extrémité du fluide. Ainsi, une distance $h$ est fixée afin que les particules qui se trouvent à une distance supérieur à $h$ ne possèdent pas d'influence sur la particule. Ainsi, on peut définir une équation 3.7 qui décrit comment une propriété physique $A_S$ à un point donné du fluide est calculée grâce aux propriétés physiques $A_i$ des particules avoisinant le point donné, pondéré par le noyau de lissage $W$ donnant l'influence des particules avoisinantes, comme nous pouvons le voir sur la figure 3.1.

$$A_S(r) = \sum_i m_i \frac{A_i}{\rho_i} W(r - r_i, h) \tag{3.7}$$

avec:

- $m_i$ la masse de la particule $i$
- $A_i$ une propriété de la particule $i$
- $\rho_i$ la densité de la particule $i$
- $W$ le noyau utilisé pour l'interpolation
- $h$ le rayon d'influence du noyau d'interpolation. Ainsi, on a $W = 0$ si $|r - r_i| > h$.

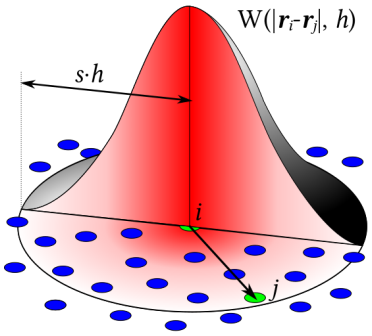



Figure 3.1: créée par Jlcercos [11]

La méthode SPH donne donc les propriétés physiques d'un point donné par une combinaison des propriétés des particules voisines. Dans notre cas, la méthode SPH résoud l'équation de la conservation de la masse 2.1 et l'équation de la conservation de la quantité de mouvement 2.4, mais pas l'équation de la conservation de l'énergie 2.5, car la température n'est pas prise en compte.

Pour plus de précisions, consultez le travail de thèse de Fabien Caleyron Caleyron [3], mais également le travail de master de Marcus Vesterlund Vesterlund [24] et le travail de thèse de Alban Vergnaud Vergnaud [23]. Cette partie a été écrite principalement avec les informations données par Vesterlund [24] et Vergnaud [23].

## 3.4 Avantages

Les principaux avantages de la méthode SPH sont que celle-ci est une méthode

- sans maillage: en effet, cela permet à la méthode SPH de simuler des problèmes avec une dynamique de frontière complexe tel que des fluides à surface libre ou des fluides avec beaucoup de déplacement de frontières. De plus, l'avantage de la méthode sans maillage est sa facilité d'implémentation et surtout de parallélisation. En effet, il est plus facile de séparer le calcul entre plusieurs unités de calcul, car on regarde en chaque point du fluide quels sont ses caractéristiques locales.
- Comme nous l'avons mentionné plus haut, la méthode SPH peut également être utilisée d'une manière plus générale comme pour la simulation d'impact à haute vitesse sur un matériaux ou encore pour la simulation d'explosions ou de fragmentations.
- qui permet une bonne conservation de la masse… (À développer ???)

## 3.5 Désavantages

Cependant, la méthode SPH possède également des inconvénients. En effet, cette méthode n'est pas bien définie aux frontières, ce qui rend la méthode difficile à calculer, comme le faisait remarquer Shadloo, Oger et Le Touzé [22]. Mais des travaux ont été faits afin de palier à ce problème, tel que Adami, Hu et Adams [1] et Kostorz [14].

(À compléter !!!)

# Chapitre 4

# Méthode LBM

## 4.1 Histoire

La “Lattice Boltzman Method” (LBM) est basée sur la méthode “Lattice Gaz Automata” (LGA) qui a été développée par J. Hardy, Y. Pomeau et O. de Pazzis [9] en 1973 sous le nom de méthode HPP, qui était basée non pas sur les équations de Navier-Stokes 2.3, mais sur les équations de Boltzmann 2.4.

Cependant, en 1986, U. Frisch, B. Hasslacher et Y. Pomeau ont réussi à obtenir les équations de Naiver-Stokes à partir de la méthode LGA en utilisant une lattice hexagonal [7]. La méthode LGA possédait des problèmes, notamment un bruit statistique causé par les approximations de la méthode LGA.

Pour pallier à ce problème, G. McMamara et G. Zanetti ont décidé de négliger les corrélations entre les particules et ont donné la notion de fonction de distribution moyenne dans leur travail [19], ce qui a donné naissance à la méthode LBM. Puis en 1989, Higuera et Jimenez ont simplifié la méthode LBM en définissant un opérateur de collision linéaire [10]. Et enfin, un opérateur de collision basé sur le travail de Bhatnagar, Gross et Krook [2] a été utilisé par simultanément par Koelman [12] et Chen et al. [4].

De nos jours, la méthode LBM est devenue populaire car elle présente une nouvelle approche efficace dans le domaine “Computational Fluid Dynamics” (CFD) qui ne consiste pas à tenter de résoudre directement les équations de Navier-Stokes 2.3, comme par exemple la méthode SPH.

### 4.1.1 Principes de base

Comme expliqué dans la section 2, un fluide est composé de plusieurs particules liées entre elles par des liaisons faibles.

La méthode LBM considère que le fluide peut être représenté par un ensemble de populations, qui sont des groupes de particules comme nous montre la figure 4.1.

Ainsi, ce ne sont pas des particules, mais des groupes de particules qui sont représentés, c’est pourquoi la physique appliquée ne sera pas déterministe, mais probabiliste.

Comme expliqué dans la section 4.1, la méthode LBM ne repose pas sur les équations de Navier-Stokes 2.3, mais sur l’équation de Boltzmann 2.4, utilisée et adaptée pour résoudre les équations de Navier-Stokes 2.3. Cependant, la méthode LBM n’est pas un solveur pour l’équation de Boltzmann 2.4.

## 4.2 Représentation et fonction de distribution

Dans la méthode LBM, le système est représenté comme un ensemble de particules dans une grille ou une lattice, comme nous le montre la figure 4.1. Le but est alors de savoir comment vont évoluer ces particules dans la grille, comment celles-ci vont interagir entre elles, quel est leurs mouvements moyen dans le système et quel est leur vitesse.

Dans les équations de Navier-Stokes 2.3, il n’y a que 2 variables qui sont la position et le temps, tandis que dans la méthode LBM, la vélocité des molécules est également perçue comme une variable qui pourra

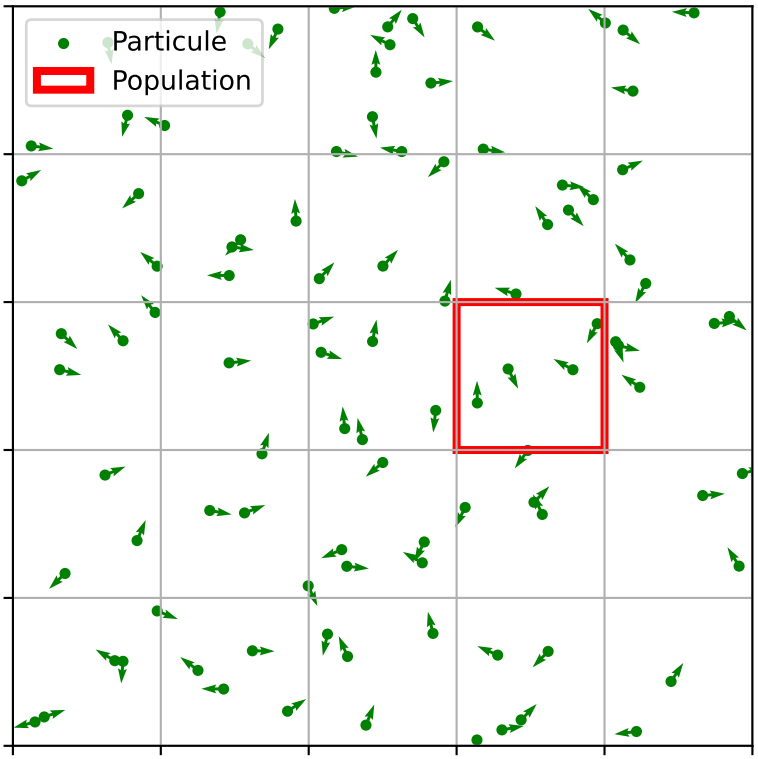



Figure 4.1: Représentation des populations de particule

être nommée $\xi$ et qui décrit le changement de position $\mathbf{x}$ de la molécule en fonction du temps $t$, comme nous le montre l'équation 4.1.

$$\xi = \frac{d\mathbf{x}}{dt} \tag{4.1}$$

Comme dans l'équation de Boltzmann 2.4, la fonction de distribution $f(\xi, \mathbf{x}, t)$ donne le nombre de molécules à une position $\mathbf{x}$ et à un temps $t$ qui se déplacent avec une vitesse $\|\xi\|_2$ dans une direction $\xi$.

La fonction de distribution $f$ est normalisée afin que l'intégration sur la vitesse et la position donne la masse du système, comme nous le montre l'équation 4.2.

$$\int d^3\xi \int d^3\mathbf{x} f(\xi, \mathbf{x}, t) = M(t) \tag{4.2}$$

### 4.2.1 Moments de la fonction de distribution

Il est possible de calculer les moments de la distribution $f$.

Ce qui est tout particulièrement intéressant est de calculer les moments de la distribution $f$ en fonction de la vitesse des molécules $\xi$ car cela permet de relier la description à l'échelle des molécules, c'est à dire de la description mésoscopique, à la description macroscopique du fluide tel que la densité, la pression ou encore la vitesse du fluide. En effet, cela permet de prendre les contributions de chaque molécule afin d'obtenir les propriétés globales du fluide.

#### 4.2.1.1 Moment d'ordre 0

Le moment d'ordre 0 par rapport à $\xi$ peut se calculer comme dans l'équation 4.3. Il permet d'obtenir la densité $\rho$ du fluide à une position $\mathbf{x}$ et à un temps $t$ car il résulte de l'intégration en fonction de la vitesse des particules que le résultat consiste à prendre chaque particule sans considérer la vitesse que peuvent avoir ces particules ce qui donne la densité de particules en une position $\mathbf{x}$ pour un temps $t$.

$$\int d^3\xi f(\xi, \mathbf{x}, t) = \rho(\mathbf{x}, t) \tag{4.3}$$

#### 4.2.1.2 Moment d'ordre 1

Le moment d'ordre 1 par rapport à $\xi$ de la fonction de distribution $f$ peut s'obtenir comme montrée dans l'équation 4.4.

$$\int d^3\xi\xi \cdot f(\xi, \mathbf{x}, t) = \rho u(\mathbf{x}, t) \tag{4.4}$$

Dans l'équation 4.4, la fonction $u$ est une fonction donnant la vitesse globale du fluide à une position $\mathbf{x}$ et à un temps $t$. En d'autres termes, cela donne la vitesse macroscopique du fluide en une position et un temps donnée.

La quantité de mouvement est alors obtenue par l'équation 4.4 car elle donne la multiplication de la densité avec la vitesse.

#### 4.2.1.3 Moment d'ordre 2

La pression du fluide est liée au moment d'ordre 2 de la fonction de distribution, ce qui ne vas pas être détaillé ici par soucis de simplicité.

## 4.3 Évolution de la fonction de distribution en fonction du temps

La fonction de densité $f$ contient toutes les propriétés locales du fluide.

Mais comment la fonction $f$ évolue-t-elle au cours du temps ?

Afin de répondre à cette question, il suffit d'observer la dérivée totale de $f$ en fonction du temps $t$, ce qui est calculé dans l'équation 4.5 grâce aux règles de chaînage et aux définitions de la vitesse $\xi$ 4.1

$$\begin{aligned}
\frac{df(\xi, \mathbf{x}, t)}{dt} &= \left(\frac{dt}{dt} \cdot \frac{\partial}{\partial t} + \frac{d\mathbf{x}}{dt} \cdot \frac{\partial}{\partial \mathbf{x}} + \frac{d\xi}{dt} \cdot \frac{\partial}{\partial \xi}\right) f(\xi, \mathbf{x}, t) \\
&= \left(\frac{\partial}{\partial t} + \xi \cdot \frac{\partial}{\partial \mathbf{x}} + \frac{F}{\rho} \cdot \frac{\partial}{\partial \xi}\right) f(\xi, \mathbf{x}, t) \\
&= \frac{\partial}{\partial t} f + \xi \cdot \nabla_{\mathbf{x}} f + \frac{F}{\rho} \cdot \nabla_{\xi} f
\end{aligned} \tag{4.5}$$

Dans l'équation 4.5, la force $F$ est appliquée sur une masse volumique $\rho$ plutôt que sur la masse $m$ de chaque particule. C'est pourquoi, grâce à la relation entre la force, la masse et l'accélération de la 2ème loi de Newton, on obtient le terme $\frac{F}{\rho}$.

### 4.3.1 Opérateur de collision $\Omega$

Comme expliqué dans la section 2.4.2, lorsque les collisions ne sont pas négligées, la variation de la fonction de distribution $f$ est donnée par les collisions entre les particules. C'est pourquoi un opérateur de collision $\Omega$ est défini.

Décrire précisément les collisions entre les particules est une chose difficile et coûteuse.

Cependant, grâce au travail de Bhatnagar, Gross et Krook (BGK) [2], comme le système va tendre vers un état équilibré $f^{eq}$ à partir de l'état non équilibré $f$, l'opérateur de collision peut être simplifié en la relaxation de $f$ vers $f^{eq}$ comme l'équation 4.6 montre, avec $\tau$ le temps de relaxation.

$$\Omega(f) = -\frac{1}{\tau}(f - f^{eq}) \tag{4.6}$$

La simplification donnée par l'équation 4.6 est suffisante pour obtenir le comportement des équations de Navier-Stokes 2.3.

## 4.4 Computation de la méthode LBM

Dans cette partie, $\mathbf{v} = \xi - u$ est considérée comme la vitesse du centre de masse d'une population de particules et les collisions entre particules sont considérées comme étant élastiques ce qui permet de conserver la masse, le moment et l'énergie.

### 4.4.1 Distribution d'équilibre $f^{eq}$

La distribution d'équilibre $f^{eq}$ doit dépendre de la densité $\rho$, de la vitesse du centre de masse $\mathbf{v}$ et de la température $T$ qui donne le degré d'agitation des particules. De plus, elle ne doit pas dépendre de la direction de la vitesse $\mathbf{v}$, donc elle doit être isotropique. Après des dérivations, la distribution d'équilibre est donnée par l'équation 4.7, qui est une simplification de la distribution de Maxwell-Boltzmann donnant la distribution des vitesses des particules dans un gaz à l'équilibre thermique.

$$f^{eq}(\mathbf{v}, \mathbf{x}, t) = \rho \left( \frac{1}{2\pi RT} \right)^{3/2} e^{-\frac{|\mathbf{v}|^2}{2RT}} \tag{4.7}$$

### 4.4.2 Discrétisation

Il ne reste plus qu'à discrétiser la fonction de distribution $f$. La discrétisation donne:

- $\mathbf{x} \to \Delta\mathbf{x}$ avec $\Delta\mathbf{x}$ qui est l'espace occupé par une population. Ainsi, l'espace est divisé en plusieurs noeuds, avec chaque noeud représentant l'espace occupé par une population. Par exemple dans la figure 4.1, $\Delta x$ sera un carré de la grille.
- $t \to \Delta t$ avec $\Delta t$ qui est en général un intervale de temps fixé, donc qui est constant, bien qu'il soit possible de modifier dynamiquement $\Delta t$.
- $f(\mathbf{v}, \mathbf{x}, t) \to f_i(\mathbf{x}, t)$ pour la vélocité $\mathbf{v}$. En fait, $\mathbf{v}$ est un vecteur. Il peut donc prendre une infinité de directions et de valeurs. C'est pourquoi, la discrétisation de la vélocité consiste à déterminer un nombre limité de directions et de valeurs que peut prendre le vecteur vitesse $\mathbf{v}$. Par exemple, la figure 4.2 illustre les vélocités que peut prendre une population. La notation D2Q9 indique que l'espace possède 2 dimensions et que seulement 9 valeurs de vélocité sont possible pour une population. $c_i$ est alors défini comme dans l'équation 4.8 ce qui permet de définir $q$ populations $f_i(\mathbf{x}, t)$, une pour chaque $c_i$ et de donner les équations 4.9 et 4.10 qui sont simplement une somme de quelques termes obtenus grâce à un mécanisme mathématique appelé "Hermite expansion".

$$(c_i) = \begin{bmatrix} 0 & 1 & 0 & -1 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 1 & 0 & -1 & 1 & 1 & -1 & -1 \end{bmatrix} \frac{\Delta\mathbf{x}}{\Delta t} \tag{4.8}$$

$$\sum_i^{q-1} f_i(\mathbf{x}, t) = \rho(\mathbf{x}, t) \tag{4.9}$$

$$\sum_i^{q-1} c_i f_i(\mathbf{x}, t) = \rho u(\mathbf{x}, t) \tag{4.10}$$

  Ainsi, par cette discrétisation, les particules se déplacent soit dans un noeud voisin, soit restent sur place. Donc la discrétisation de la vélocité est parfaitement alignée sur la discrétisation du temps et de l'espace.

Les discrétisations introduites permettent de discrétiser également la fonction de distribution équilibrée comme montrée dans l'équation 4.11 avec $w_i$ le poids d'une population qui dépend de si la population se déplace le long des axes principaux ou diagonaux ou encore si elle reste sur place.

$$f_i^{eq} = w_i \rho \left( 1 + \frac{c_i \cdot u}{c_S^2} + \frac{(c_i \cdot u)^2}{2c_S^4} - \frac{u \cdot u}{2c_S^2} \right) \tag{4.11}$$

La variable $c_S$ représente la vitesse du son et est définie dans l'équation 4.12.

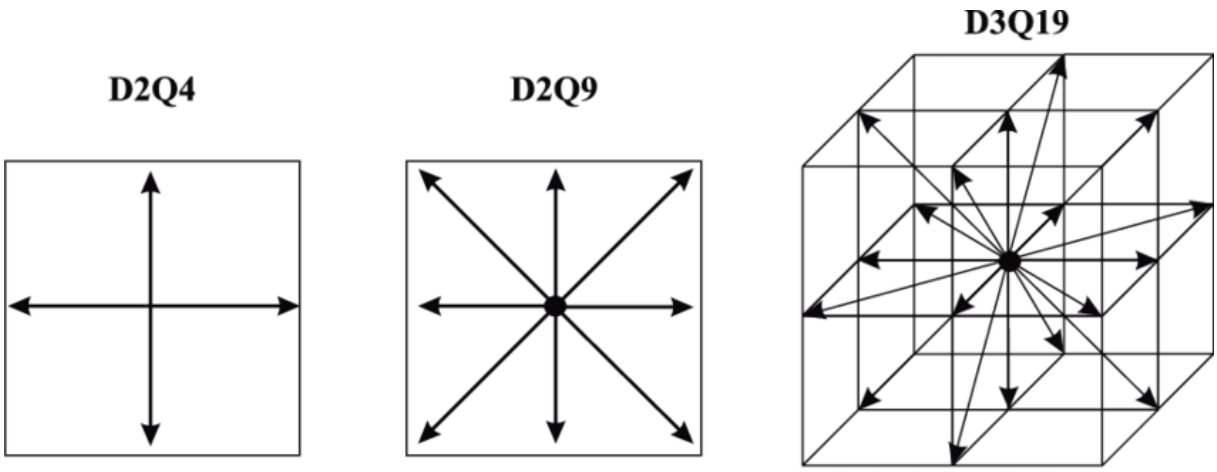



Figure 4.2: prise du travail de Körner et al. [13]

$$c_S = \frac{1}{\sqrt{3}} \frac{\Delta \mathbf{x}}{\Delta t} \tag{4.12}$$

### 4.4.3 Lattice Boltzmann Equation (LBE)

L'objectif est de savoir pour une fonction de distribution $f$ donnée comment celle-ci évoluera au temps $t + \Delta t$.

La "Lattice Boltzmann Equation" (LBE) ou "Lattice BGK equation" est alors définie comme dans l'équation 4.13.

$$\frac{df}{dt} \approx f_i(\mathbf{x} + c_i \Delta t, t + \Delta t) - f_i(\mathbf{x}, t) = -\frac{\Delta t}{\tau}(f_i(\mathbf{x}, t) - f_i^{eq}(\mathbf{x}, t)) = \Omega_i \tag{4.13}$$

Ce qui peut se réécrire comme l'équation 4.14.

$$f_i(\mathbf{x} + c_i \Delta t, t + \Delta t) = f_i(\mathbf{x}, t) - \frac{\Delta t}{\tau}(f_i(\mathbf{x}, t) - f_i^{eq}(\mathbf{x}, t)) \tag{4.14}$$

Dans l'équation 4.14, le terme de relaxation donne un déplacement ou une redistribution de la population $i$, qui ne s'est pas encore déplacée. Elle est ensuite propagée vers ses voisins.

Les avantages sont que les collisions sont locales et algébriques: c'est un algorithme très simple pour les collisions et la propagation est linéaire et exacte, ce qui signifie que l'on peut décomposer le solveur 4.14 en étape de collision et de propagation.

### 4.4.4 Viscosité

Les équations de Navier-Stokes 2.3 utilisent le terme de viscosité tandis que l'équation de Boltzmann 2.4 utilise le terme de temps de relaxation.

Les 2 termes sont liés par l'équation 4.15 avec la vitesse du son $c_S$ définie dans l'équation 4.12.

$$\Sigma = c_S^2 \left(\tau - \frac{\Delta t}{2}\right) \tag{4.15}$$

### 4.4.5 Algorithme

- La condition initiale $f_i$ est connue. Ainsi, il est possible de calculer la densité et la vélocité par la somme des moments de $f$.

- La distribution équilibrée peut donc être calculée grâce à la condition initiale $f_i$ car elle ne dépend que de la densité et de la vélocité, qui ont pu être calculé précédement.
- La distribution post collision $f_i^*(\mathbf{x},t) = f_i(\mathbf{x},t) - \frac{\Delta t}{\tau}(f_i(\mathbf{x},t) - f_i^{eq}(\mathbf{x},t))$ est ensuite calculée grâce à la distribution équilibrée et à la condition initiale car le temps de relaxation est fixé afin d'avoir une viscosité donnée. Elle donne comment les populations sont redistribuées dans l'espace des vélocités, mais ne déplace pas les populations. Il s'agit de l'étape collision de la figure 4.3.
- La propagation de la distribution post collision $f_i^*$ aux voisins: $f_i(x + c_i\Delta t, t + \Delta t) = f_i^* i(\mathbf{x},t)$. C'est l'étape propagation de la figure 4.3. Cela est représenté dans la figure 4.3.

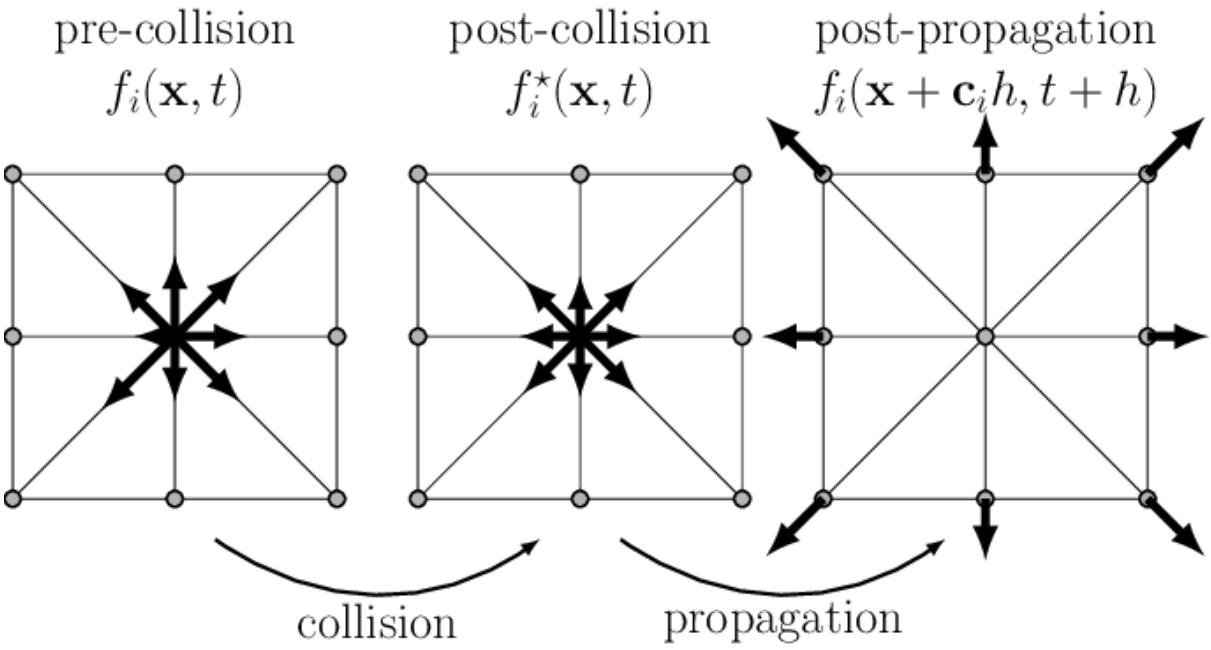



Figure 4.3: prise dans le travail de Schiller et al. [21]

Pourquoi ça marche ?

- On a une "lattice symetry and isotropy"
- La masse et la quantité de mouvement sont conservées car: $\sum_i \Omega_i = 0$ et $\sum_i c_i \Omega_i = 0$
- Les analyses de Champman-Enskog [**Champman**] et [6] montrent que les LBE 4.4.3 sont suffisantes pour retrouver le comportement de Navier-Stokes.

## 4.5 Avantages

- La méthode LBM est rapide car elle est explicite et linéaire en propagation.
- Il n'y a pas d'équation de poisson à résoudre car on ne résoud pas directement les équations de Navier-Stokes 2.3.
- La méthode LBM est locale.
- Le fait que la méthode LBM est locale la rend facilement parallélisable car c'est possible de séparer le calcul. De plus, la méthode LBM est très scalable.
- Les géométries complexes peuvent être implémentées facilement

## 4.6 Désavantages

- Il n'est possible de simuler que des petits nombres de Knudsen. Le nombre de Knudsen décrit le régime d'écoulement d'un fluide. Lorsqu'il est grand, cela signifie que les collisions entre les particules sont rares, à l'inverse de lorsqu'il est petit. Donc il n'est possible de simuler que de la mécanique des fluides classique où les collisions entre les particules ne sont pas rares.
- Il n'est également pas possible de simuler des grands nombres de Mach. Le nombre de Mach représente un rapport entre la vitesse du flux d'un fluide et la vitesse du son dans le fluide. Donc un petit nombre de Mach signifie que la vitesse du flux du fluide est faible par rapport à la vitesse du

son, ce qui caractérise des écoulements incompressibles ou à faible compressibilité. À l'inverse, un grand nombre de Mach représente les fluides où les effets de compressibilité sont important. Il n'est donc possible de simuler que des fluides incompressibles ou faiblement compressibles.
- Les conditions aux bords doivent correctement être définies. En effet, les conditions aux bords définiront la cohérence et la précision du résultat.

Ces explications ont essentiellement été tirées du travail du Dr. Moritz Lehmann [16], qui est l'auteur de FluidX3D, mais également du travail et des explications de Krüeger [**Krueger**] et [15].

# partie II

# Résultats

# Chapitre 5

# La containerisation

Dans ce rapport, beaucoup de théorie a déjà été faite sur différentes méthodes de simulation. Cela est dû en grande partie parce que je considère qu'il est important de savoir comment ça marche avant de comparer les méthodes, et parce que j'ai trouvé la méthodologie très intéressante.

Je me permettrait d'utiliser un jargon informatique dans ce chapitre pour plus de clarté. En effet, un jour ma mère m'a demandé ce que j'avais fait, et je lui ai répondu "J'ai enlevé des insectes de ma poubelle" ce qui signifie en jargon informatique "j'ai enlevé des bugs dans mon container"

Dans ce chapitre, je vais un peu expliquer la containerisation car cela a fait grandement partie de mon projet de bachelor.

En effet, j'ai dû reprendre une application qui datait de 2017 et qui ne fonctionnait plus car elle n'était plus compatible avec les librairies actuelles et réussir à installer la version de 2017 des librairies est compliqué (je n'y suis pas parvenu).

C'est à cet instant que la containerisation prend toute son importance.

## 5.1 Notions générales

Un ordinateur ne comprend que des zéros et des uns, ce qui n'est pas le cas de l'homme. C'est pourquoi, il y a toujours un système d'exploitation qui tourne sur l'ordinateur afin de servir d'interface entre l'homme et la machine. Un système d'exploitation est composé d'un noyau ou kernel, d'outils systèmes et de librairies permettant à l'utilisateur d'interagir avec la machine, contrôlée par le noyau.

Une machine virtuelle (VM) est une émulation d'un système informatique physique, créée par un logiciel de virtualisation. Elle permet d'exécuter un système d'exploitation complet et ses applications sur un matériel virtuel, indépendamment du matériel physique sous-jacent. Les machines virtuelles offrent une isolation forte, une grande flexibilité et sont largement utilisées dans diverses applications informatiques (ChatGPT).

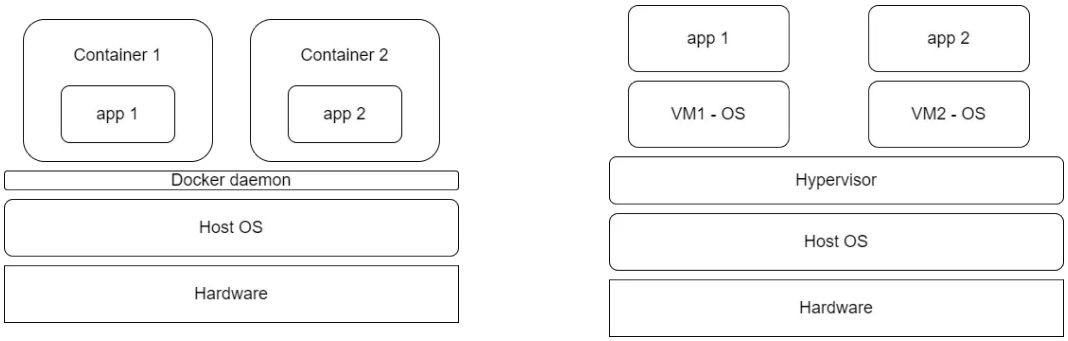
Les machines virtuelles sont gérées par un hyperviseur, comme il est possible de voir sur la figure 5.1b, qui s'occupe alors notamment de l'isolation et de la gestion des ressources des machines virtuelles.

Ainsi, il est possible de mettre une application sur une machine virtuelle car toutes ses dépendances pourront être installées sur le système d'exploitation du matériel virtuel. L'application pourra alors fonctionner sur n'importe quel matériel physique possédant un hyperviseur pour machine virtuelles.

Cependant, un système d'exploitation occupe beaucoup de mémoire. Par exemple, Windows 10 occupe environ 20Go de mémoire. Donc est-ce nécessaire de déployer une machine virtuelle pour une application qui fait seulement 10Mo ? Ou alors c'est une perte de place ? Et est-ce que c'est vraiment utile d'avoir toutes les fonctionnalités du système d'exploitation pour déployer l'application, ou alors on peut se contenter de seulement quelques fonctionnalités ?

Les containers sont alors une solution à ce problème. En effet, un container ne virtualise non pas le matériel (comme le stockage ou la mémoire), mais un système d'exploitation, ce qui signifie qu'ils n'ont pas besoin d'avoir leur propre système d'exploitation, ce qui fait notamment un grand gain de stockage.

Cela permet également de n'utiliser que les fonctionnalités du système d'exploitation qui sont nécessaire au bon fonctionnement du container. Il en résulte que les containers ont un impact beaucoup plus faible sur les performances du matériel physique qu'une machine virtuelle. Le principe du container est simple: mettre dedans le code et toutes ses dépendances afin de pouvoir l'exécuter sur n'importe quel système. Les containers sont gérés par un démon, comme il est possible de voir sur la figure 5.1a.




(a) Container

(b) VM

Figure 5.1: Virtual Machine (VM) et container [20]

## 5.2 Pourquoi utiliser la containerisation ?

Dans mon projet de bachelor, je me suis occupé de rendre fonctionnel un code qui n'a pas eu de maintenance depuis 2018 environ.

J'ai donc dû comprendre ce qui ne fonctionnait pas et le rendre exécutable, ce qui fut résolu avec succès. Ici, si l'auteur du code avait créé un container, je n'aurais pas eu besoin de devoir mettre à jour le code, ce qui est beaucoup plus pratique.

À ce moment, je ne m'étais toujours pas penché sur la containerisation.

Cependant, le projet avançant, je me suis heurté à des comportements spéciaux de mon code, et comme celui-ci utilise une carte graphique NVIDIA, je ne pouvais pas montrer à mon professeur encadrant mes problèmes afin d'avoir une piste de déblocage. Mais mon professeur possède des serveurs distants équipés de cartes graphiques NVIDIA.

On a donc essayé d'exécuter le code sur le serveur distant, mais on s'est heurté à des problèmes de librairies manquantes et d'autres problème.

C'est alors que la containerisation a pris toute son importance dans mon projet de bachelor.

En effet, l'objectif était alors de mettre le code dans un container qui s'exécutera sur le serveur distant du professeur afin de lui montrer mes problèmes et de pouvoir directement essayer des solutions.

L'idée est d'exécuter un container et d'afficher le résultat graphique du container.

Pour cela, il faut comprendre les mécanismes régissant l'affichage X11.

## 5.3 Affichage X11

L'affichage X11 est composé notamment de:

- Un serveur Xorg ou X11 qui gère l'affichage et les périphériques d'entrée tel que la souris et le clavier. Lors de son démarrage, le serveur X11 crée un socket permettant la communication inter-processus sans passer par le réseau qui va être à l'écoute des connections des applications clientes, et un fichier de stockage de "magic cookies", généralement appelé `.Xauthority`, afin d'identifier les clients souhaitant afficher quelque chose. Le chemin d'accès au fichier `.Xauthority` est mis dans la variable $XAUTHORITY. Lorsqu'une application voudra afficher quelque chose, elle se connectera au socket créé par le serveur et discutera avec celui-ci grâce au socket.

- Des “magic cookies” qui sont des jetons d’authentifications permettant d’accéder au serveur X11. Ils utilisent le hostname, le type de connection et le numéro d’affichage afin de composer un identifiant pour le jeton d’accès. Dans l’exemple de “magic cookies” suivant, lizzy est le hostname de mon ordinateur, combiné avec le type de connection qui est unix et le numéro d’affichage.

```
lizzy/unix:0  MIT-MAGIC-COOKIE-1  142d43ebe5215cfc2ae826e4c592de8e
```

Lorsque le client s’est connecté au socket, il doit s’authentifier auprès du serveur X11, ce qui est fait par l’extraction du “magic cookie” du fichier `.Xauthority` et par l’envoi de ce cookie au serveur. Le serveur décide alors si le client a les permissions d’afficher ou pas. - Une variable d’environnement ´$DISPLAY´ utilisée afin d’indiquer à l’application cliente quel socket utiliser. La variable d’environnement est composée de la sorte: `hostname:numéro_d_affichage.numéro_du_serveur_X11`

- Le hostname indique le nom du host où se trouve le serveur X11.
- Le numéro d’affichage indique où l’application devra être affichée.
- Le numéro du serveur X11 est utile s’il y a plusieurs serveurs X11 qui tournent sur une machine.

Il est possible que la variable `$DISPLAY` soit sous la forme `:0.0`. Dans ces conditions, cela signifie que l’affichage se fera sur le serveur X11 local, ce qui peut également s’écrire comme `localhost:0.0`.

La variable d’environnement permet d’indiquer au client le socket à utiliser.

## 5.4 Affichage d’un container sur ordinateur

On peut faire tourner toute sorte de code dans un container, notamment du code donnant un rendu visuel. La difficulté consiste alors à afficher le rendu visuel sur l’écran.

Mais grâce à la section 5.3, nous savons comment fonctionne en gros le serveur X11 responsable de l’affichage à l’écran.

Donc pour afficher le rendu visuel du code, il faut donner lors de l’exécution du container la variable d’environnement `$DISPLAY` pour que le container sache quel socket utiliser, il faut également partager le socket responsable de la connection avec le serveur et le fichier contenant les cookies d’authentification, ainsi que le même hostname que l’ordinateur sur lequel tourne le container afin que celui-ci puisse trouver le bon cookie à envoyer au serveur. Cela est illustré par la figure 5.2.

Il se peut toutefois que le fichier contenant les cookies n’existe pas. C’est pourquoi, il faut d’abord penser à le créer, puis à créer les cookies.

Cela donne avec docker:

```
# Vérifier que .Xauthority existe
# et le créer avec les cookies du display actuel le cas échéant
if [ ! -e $HOME/.Xauthority ] then
  touch $HOME/.Xauthority
  xauth generate $DISPLAY . trusted
  export XAUTHORITY=$HOME/.Xauthority
fi

docker run --gpus all --runtime nvidia \
  -e DISPLAY=$DISPLAY \ # Partage de la variable $DISPLAY
  -e XAUTHORITY=/home/user/.Xauthority \ # Indication de la location de .Xauthority
  -v /tmp/.X11-unix:/tmp/.X11-unix \ # Partage du socket X11
  -v $XAUTHORITY:/home/user/.Xauthority \ # Partage des cookies
  --hostname $(hostname) \ # Partage du hostname
  mon-application # Image de l'application graphique
```

## 5.5 Affichage d’un container sur serveur distant

Le fait de pouvoir avoir le rendu visuel d’un container docker est une bonne chose, mais il fallait pouvoir avoir un rendu sur un serveur distant.

Le container s'exécute alors sur un serveur distant.

Une connection ssh avec l'option -X est alors établie afin d'afficher sur notre ordinateur le résultat.

Comme nous pouvons voir sur la figure 5.3, la connection ssh crée un socket TCP qui transfert par tunnel ssh la sortie visuelle de ce qui est exécuté sur la machine distante. Le soucis est qu'il n'est pas possible de partager ce socket à l'image docker à exécuter. C'est pourquoi, on utilise un socket unix que l'on redirige vers le socket tcp afin d'envoyer l'affichage dans le tunnel ssh, ce qui est fait au moyen le la commande `socat`. Notez ici que le socket TCP est identifié par la variable `$DISPLAY` qui est sous la forme `localhost:10` indiquant que le socket TCP écoute sur le port $6000 + 10 = 6010$ pour le transfert X11. Il faut ensuite redéfinir la variable `$DISPLAY` afin que celle-ci pointe bien sur le bon socket, c'est à dire le socket unix. L'authentification du client suit exactement le même principe que pour l'affichage sur un ordinateur 5.4.

C'est pourquoi, il est possible d'écrire le script suivant:

```
#!/bin/bash

# Prise du numéro d'affichage
DISPLAY_NUMBER=$(echo $DISPLAY | cut -d. -f1 | cut -d: -f2)

# Création de .Xauthority s'il n'existe pas
if [ ! -e $HOME/.Xauthority ]; then
  touch $HOME/.Xauthority
  xauth generate :$DISPLAY_NUMBER . trusted
  export XAUTHORITY=$HOME/.Xauthority
fi

# Test si on est sur un serveur distant
# et liaision du socket TCP de la connection ssh
# avec le socket UNIX
if echo $DISPLAY | grep -q "localhost"; then
  socat UNIX-LISTEN:/tmp/.X11-unix/X${DISPLAY_NUMBER},fork TCP4:localhost:60${DISPLAY_NUMBER} &
fi

# Lancement de l'application graphique
docker run --gpus all --runtime nvidia \
  -e DISPLAY=:$DISPLAY_NUMBER \
  -e XAUTHORITY=/home/user/.Xauthority \
  -v /tmp/.X11-unix:/tmp/.X11-unix \
  -v $XAUTHORITY:/home/user/.Xauthority \
  --hostname $(hostname) \
  mon-application
```

## 5.6 Conclusion

Cela a été fort intéressant, mais également très chronophage.

En effet, l'application ne marchait tout de même pas sur le serveur du professeur à cause d'une librairie utilisée (je crois la librairie OpenGL si je me souviens bien).

J'ai donc testé d'utiliser mon ordinateur fixe, afin de voir si le problème persiste. Pour cela, j'ai dû installer et configurer le serveur ssh sur mon ordinateur afin d'avoir une connection si possible un peu sécurisée, mais j'ai également configuré mon routeur afin d'autoriser les connections depuis l'extérieur, ce qui s'est tout de même soldé par un échec. En effet, le problème observé sur l'ordinateur du professeur persistait.

Mais comme le temps avance vite, je n'ai pas encore réussi à trouver une solution à ce problème.

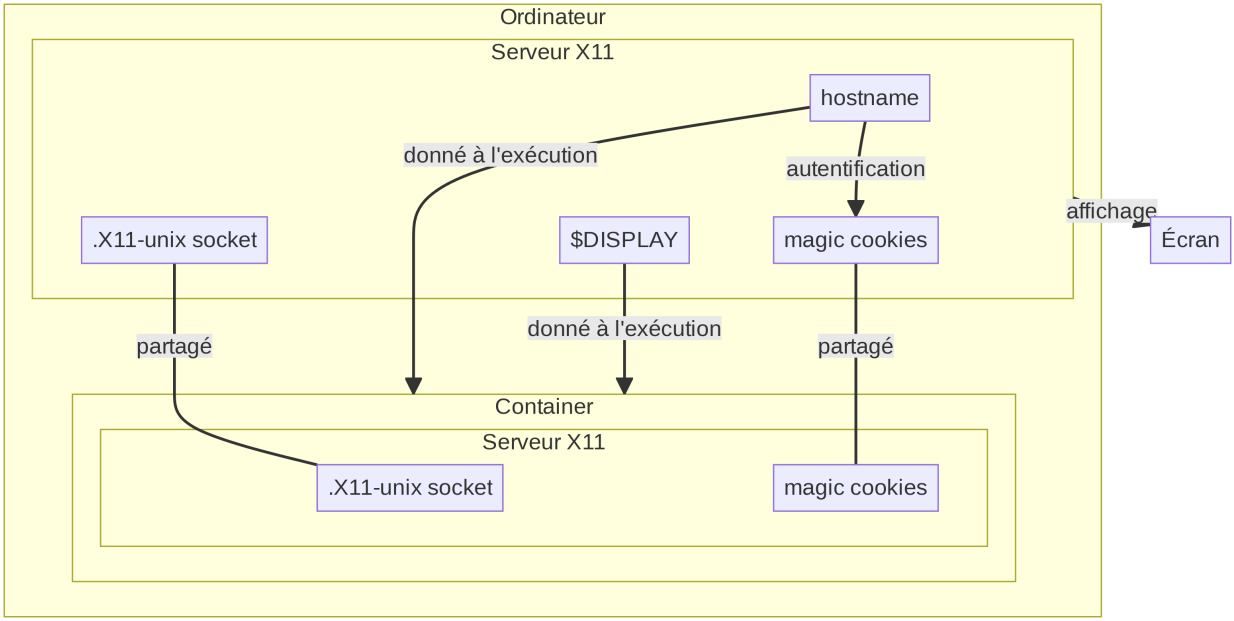



Figure 5.2: Shéma de l'affichage d'une application graphique tournant dans un container sur un ordinateur

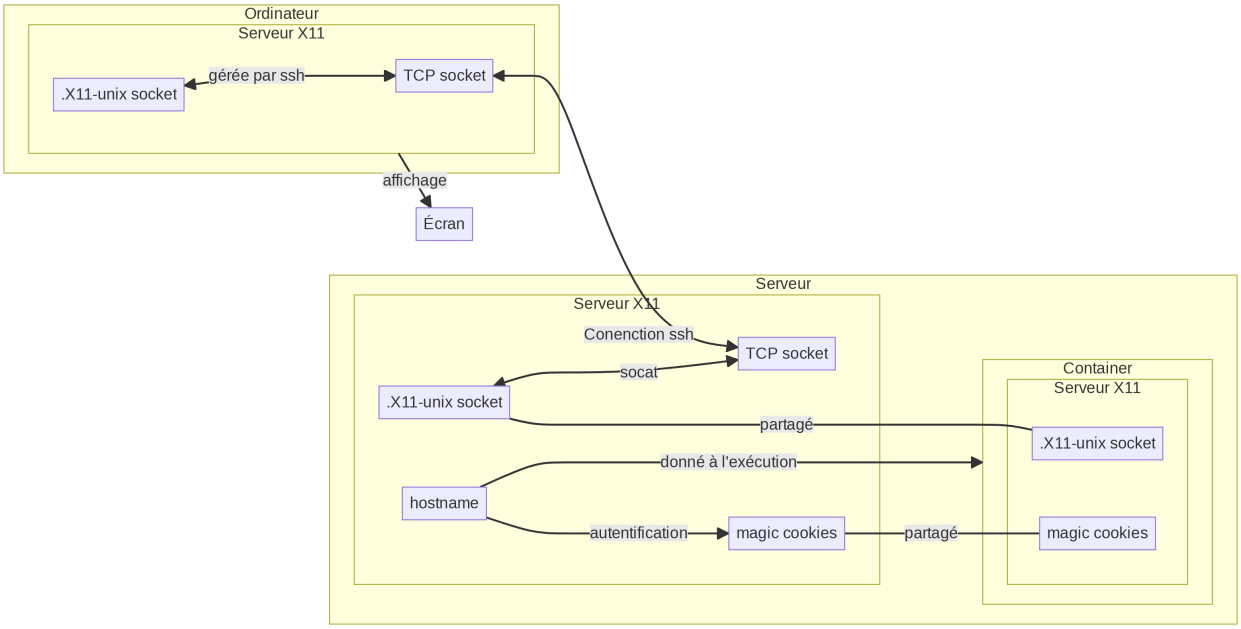



Figure 5.3: Shéma de l'affichage d'une application graphique tournant dans un container sur un serveur distant

# Chapitre 6

# LBM VS SPH

## 6.1 Introduction

Après avoir vu la théorie sur la méthode SPH 3 et la méthode LBM 4, nous pouvons procéder à une comparaison entre deux codes implémentant chacun une des méthodes.

- Le premier code est le code de GVDB-Voxel implémentant la méthode SPH
- Le deuxième code est le code de FluidX3D implémentant la méthode LBM

Comme cela avait été mentionné précédement, le code de GVDB-Voxel n'a pas été maintenu depuis au moins 4 ans, ce qui a donné des problèmes d'incompatibilité entre les versions des librairies et le code, à la différence du code de FluidX3D qui est en cours de développement par le Dr Moritz Lehmann, dont le projet de master a servi à la compréhension de la méthode LBM.

Les deux codes proposés présentent chacun des caractéristiques différentes, tant par la méthode utilisée pour simuler en temps réel des fluides que par les outils et technologies utilisées.

Dans la suite du travail, l'objectif sera de comparer ces deux codes.

## 6.2 Comparaison théorique

Le code de GVDB-Voxel implémente, comme mentionné précédement, la méthode SPH alors que le code de FluidX3D implémente la méthode LBM.

Ces deux méthodes ont une façon totalement différente d'approcher le problème de la simulation de fluides.

### 6.2.1 Approche utilisée

La méthode SPH tente une approximation directe des équations de Navier-Stokes 2.3 tandis que la méthode LBM utilise l'équation de Boltzmann 2.4 afin de retrouver le comportement des équations de Navier-Stokes 2.3, autrement dit, la méthode LBM résoud indirectement les équations de Navier-Stokes. Ce qui lui permet, à la différence de la méthode SPH, d'intégrer relativement facilement des effets thermiques grâce à l'utilisation des fonctions de distribution pour les énergies.

En utilisant l'équation de Boltzmann 2.4, la méthode LBM possède un plus grand nombre de paramètres que les méthodes résolvant directement les équations de Navier-Stokes, car la vélocité de chaque groupe de particules ou population est prise en compte. Cependant, cela permet de ne pas avoir à résoudre directement les équations de Navier-Stokes, ce qui en fait une méthode rapide car aucune équation de poisson n'est à résoudre, et cela permet également de simuler des fluides à micro-échelle où les interactions moléculaires sont significatives.

### 6.2.2 Maillage

La méthode SPH est une méthode sans maillage alors que la méthode LBM est composée d'un maillage. Cela permet à la méthode SPH de définir un domaine adaptatif qui varie en fonction des besoins de la

simulation à la différence de la méthode LBM qui possède un domaine fixe. Un autre avantage conféré à la méthode SPH par cet absence de maillage est une facilité d'adaptation à des changements topologiques comme la fragmentation ou la fusion de fluides, et une simplicité de définition des conditions aux frontières, bien que cela reste toujours très complexe afin d'obtenir de bons résultats. Cependant, la méthode LBM est également capable de gérer des frontières complexes, comme des frontières en mouvement ou des géométries complexes grâce à sa localité qui lui permet également une flexibilité sur la définition des conditions aux limites.

Mais la méthode SPH possède plus de difficultés à traiter des conditions aux limites solides car cela peut nécessiter des techniques spéciales afin de modéliser l'interaction entre le fluide et la surface solide, ce qui n'est pas très complexe dans le cas d'une méthode avec un maillage.

Il résulte néanmoins du fait que la méthode LBM soit une méthode avec maillage que la précision est dépendante du maillage choisi, bien qu'il existe des méthodes de raffinement de maillage ou de grille adaptative afin de palier à ce problème.

### 6.2.3 Phénomènes physiques

La méthode LBM perd en efficacité dans les simulations d'écoulements à haute vitesse ou d'écoulements compressibles, car des corrections doivent être faites afin de prendre en compte les écoulements haute vitesse, ce qui complexifie l'implémentation. Mais la méthode SPH n'est également pas très efficace dans les simulations d'écoulements à haute vitesse, car cela requiert un nombre plus élevé de particules, et donc de temps de calcul, de mémoire… Cependant la méthode SPH est très efficace pour simuler des phénomènes avec des variations rapides de densité et de pression, comme des explosions ou des fluides compressibles.

Comme vu précédement, la méthode SPH est également utilisée pour la simulation d'explosions, de fragmentation de matériaux, d'impact haute vitesse… La méthode LBM ne permet pas de faire cela, cependant, elle possède des applications multi-physiques tel que les écoulements multiphasiques comme les interactions entre fluides immiscibles, les interactions entre fluide et structure déformable ou rigide, les transfert de chaleur, les réactions chimiques et la combustion dans les fluides, les écoulements à petite échelle…

À la différence de la méthode SPH, la méthode LBM permet également de simuler des fluides non newtoniens.

## 6.3 Comparaison technologique

### 6.3.1 Performances

#### 6.3.1.1 Calcul parallèle

La méthode LBM, comme vu dans la section 4.4.5, permet de facilement séparer les étapes permettant de calculer le comportement du fluide.

En effet, la méthode LBM est locale et les calculs de propagation et de collision peuvent être fait localement, indépendemment, sur chaque partie du maillage.

Ainsi, il en résulte que la méthode LBM est particulièrement adaptée au calcul parallèle et hautement scalable car lorsque la puissance de calcul augmente, l'overhead, qui est une mesure de la puissance de calcul inutilisée, reste à peu près constant.

La caractéristique d'être hautement parallélisable et scallable rend cette méthode très attractive de nos jours.

La méthode SPH est également adaptée au calcul parallèle.

En effet, la méthode SPH représente le fluide comme un ensemble de particules qui interagissent entre elles. Cependant elles n'interagissent pas avec toutes les particules mais seulement avec des particules dans un certain rayon défini lors de l'initialisation de la simulation, comme l'image 3.1 peut illustrer. Ainsi, grâce à la localité des interactions entre particules, il est possible de diviser le calcul entre de nombreuses unités de calculs, ce qui signifie que la méthode SPH est également adaptée au calcul parallèle et est également scallable.

#### 6.3.1.2 Utilisation de la mémoire

L'utilisation de la mémoire est un aspect crutial dans les deux méthodes.

En effet, la méthode SPH doit stocker un grand nombre de particules possédant chacune des caractéristiques telle que sa position, sa vitesse et la méthode LBM doit stocker pour chaque noeud du maillage plusieurs distributions de vitesse.

Ainsi, l'aspect de l'utilisation de mémoire est un défi dans les deux méthodes et doit être optimisé afin d'obtenir des codes efficaces. Effectivement, si l'utilisation de la mémoire n'est pas obtimisé, il est possible que des simulations à haute résolution dépassent la capacidé de mémoire de la machine sur laquelle elle s'exécute.

Mais pour les deux codes proposés, l'utilisation de la mémoire est correctement gérée.

### 6.3.2 Technologies utilisées

Les deux implémentations des méthodes LBM et SPH sont basées sur du C++, mais elles diffèrent quant à l'utilisation des technologies.

La méthode LBM utilise la librairie OpenCL alors que la méthode SPH utilise la librairie CUDA de Nvidia. Cependant, d'après le travail [17] du Dr Moritz Lehmann, les deux librairies sont aussi rapides sur GPU Nvidia si OpenCL est correctement utilisé.

Cela signifie donc que FluidX3D possède l'avantage, comparé au code de GVDB-Voxel, de pouvoir tourner sur tous les GPUs, et d'être aussi rapide sur GPU Nvidia que s'il avait été implémenté en CUDA.

Donc de ce côté, il y a des différences, mais les deux technologies s'équivalent.

La méthode SPH est implémentée au moyen de la technologie Nvidia GVDB-Voxel. Elle utilisait OptiX pour le raytracing, qui a été supprimé en faveur du raytracing intégré de CUDA en raison de la difficulté de trouver la bonne librairie afin de faire tourner le code.

#### 6.3.2.1 GVDB-Voxel

La technologie GVDB-Voxel a été développée vers 2018 pour le calcul, la gestion, la simulation et le rendu de données volumétriques éparses à grande échelle sur des GPUs Nvidia. Elle est uniquement dépendante de CUDA.

Les données volumétriques éparses sont des données volumétriques, représentant un volume dans l'espace, qui ne sont pas uniformément denses dans l'espace, ce qui signifie qu'il peut y avoir des régions avec beaucoup de données et des régions avec peu de données.

Basée sur la technologie OpenVDB, GVDB-Voxel tente de représenter les données volumétriques éparses d'une manière efficace.

Pour cela, la technologie est basée sur une structure de grille hiérarchique.

##### 6.3.2.1.1 Grille hiérarchique

La structure de la grille hiérarchique est donnée à l'initialisation du programme. Elle déterminera le nombre de voxels total qui pourra être utilisé.

La grille est composée de:

- voxels: ce sont les plus petites unités de données volumétriques. Cela correspond à un pixel, mais en 3 dimensions.
- bricks ou blocs: ce sont des collections de voxels. On les appelle bricks ou blocs car ce sont des petit cubes de voxels, généralement de taille fixe. Les bricks rendent l'accès plus facile et plus rapide aux voxels, ce qui permet une meilleure gestion des données.
- Nodes ou noeuds: ce sont des pointeurs vers d'autres noeuds ou vers des bricks. Ils sont utilisés afin de pouvoir organiser les bricks d'une manière hiérarchique. Il peut y avoir plusieurs niveau de noeuds, ce qui permet de diviser l'espace volumétrique en régions aussi petites que l'on souhaite.

Ces composants sont illustrés dans la figure 6.1 pris de la documentation de GVDB-Voxel [5] avec en bleu les voxels, en vert les bricks et en rouge les nodes.

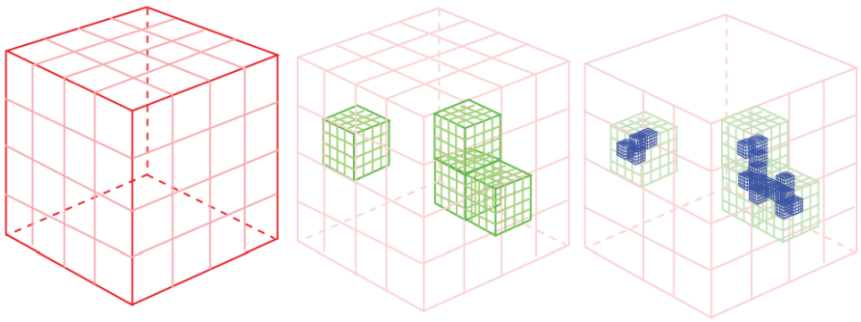


Figure 6.1: Grille hiérarchique [5]

L'implémentation concrête de cette hiérarchie peut se faire en liste de noeuds possédant une liste d'enfants qui pointent soit vers des bricks, soit vers d'autres noeuds, et ainsi de suite, comme le montre la figure 6.6.

L'objectif est maintenant de stocker toutes ces données d'une manière efficace dans la mémoire du gpu afin d'avoir des accès rapides aux données, ce qui est crucial lorsque l'on souhaite obtenir des performances car les accès à la mémoire sont très coûteux.

Pour ce faire, la technologie GVDB-Voxel utilise des atlas, qui sont des structures de données créées afin d'optimiser le stockage et l'accès aux bricks.

Un atlas est donc composé de bricks qui sont dynamiquement allouées lors de l'exécution. Il possède une taille maximale déterminée à la compilation et un type de donnée qu'il peut stocker. Les bricks sont indexés en fonction de leurs coordonnées dans l'espace, ce qui permet un accès direct et rapide aux données. Par exemple, si un voxel possédant des coordonnées données est recherché, il est possible de trouver directement l'index de l'atlas dans lequel le voxel se trouve grâce à un masque.

Comme le montre la figure 6.5, un atlas stocke les bricks d'une manière contigüe en mémoire, ce qui rend les données facilement gérables et accessibles et les transferts de donnée plus facile et rapide, par exemple entre le CPU et le GPU lors de calculs.

De plus, lorsque des bricks sont désallouées, l'atlas peut directement réutiliser l'espace laissé vide pour d'autres bricks, ce qui optimise la quantité de mémoire utilisée.

Comme le montre la figure 6.2, l'atlas ne stocke pas seulement les bricks, mais également les apron voxels qui sont les voxels voisins de la bricks stockée afin d'éviter les communications entre les unités computationnelles lors d'un calcul parallèle nécessitant des calculs de voisinage, car les données dont ils ont besoin se trouvent dans les apron voxels.

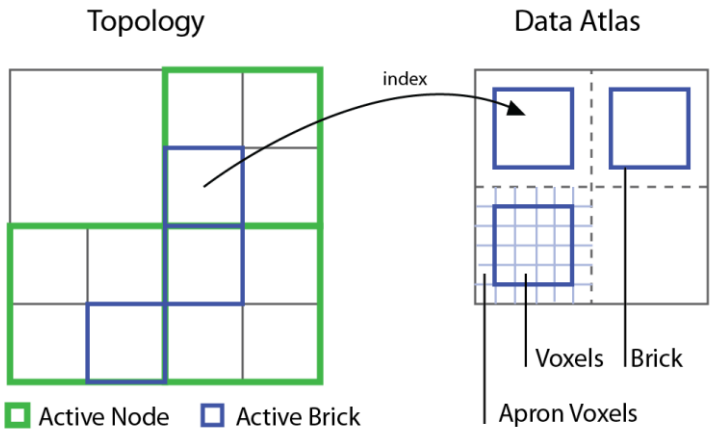



Figure 6.2: Atlas et stockage des bricks [5]

Ainsi, peu de communications entre les unités de calculs sont nécessaires, mais peu de données sont stockées, à la différence des données volumétriques denses qui permettent un calcul rapide avec accès

facile aux voisins d'un voxel, mais qui n'optimisent pas du tout la quantité de mémoire utilisée, comme le montre la figure 6.3.

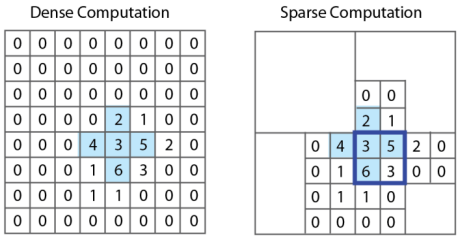



Figure 6.3: Différence entre donnée volumétrique éparse et dense lors d'un calcul [5]

Les atlas peuvent être utilisés pour stocker différentes informations, telle que les couleurs ou la densité, comme la figure 6.4 le montre.

Ainsi, l'accès aux informations est rapide et facile. Par exemple, si un besoin de connaître la densité de telle région du volume se présente, il suffit de chercher dans l'atlas correspondant au canal densité à l'index donné par les coordonnées de la région étudiée appliqué à un masque car toute la coordonnée n'est pas nécessaire pour localiser la bricks, sinon il n'y aurait aucune utilité à créer des bricks qui sont des groupes de voxels.

En résumé, un atlas est un espace de stockage de bricks gérant efficacement la mémoire, facilitant l'accès aux données et permettant d'éviter des communications entre les unités computationnelles lors de calculs nécessitant les informations de voisinage.

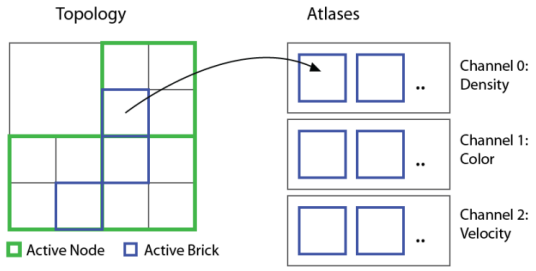



Figure 6.4: Multiples atlas pour stocker des canaux [5]

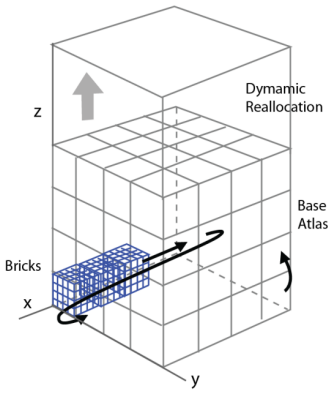



Figure 6.5: Shéma représentant un atlas

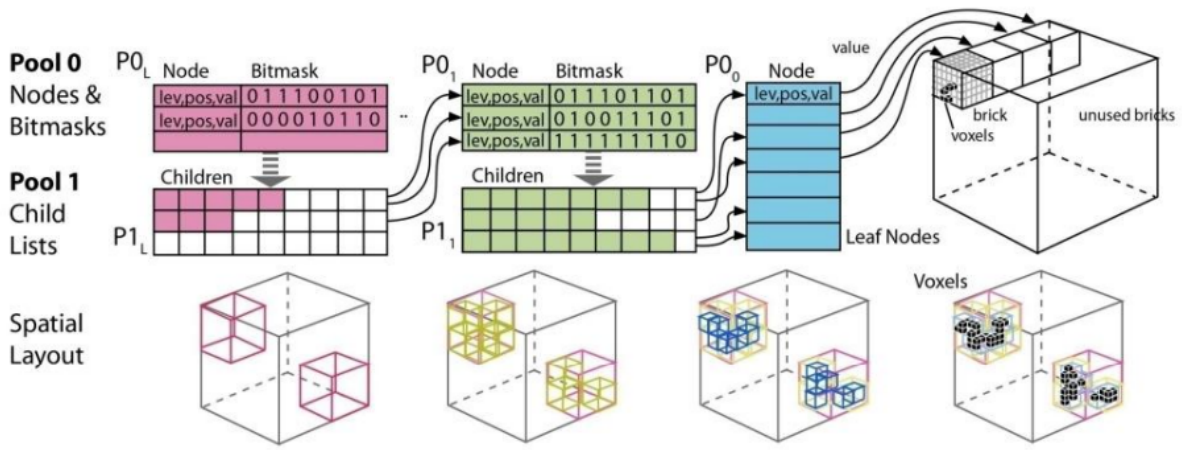



Figure 6.6: Représentation de l'implémentation de la structure de grille hiérarchique

# Chapitre 7

# Comparaison entre les résultats

Dans ce chapitre, une comparaison entre les deux codes est proposée. Afin de faire un petit rappel, on a comme code:

- GVDB-Voxel: ce code est une implémentation de la méthode SPH en tirant profil des technologies de Nvidia
- FluidX3D: ce code est une implémentation de la méthode LBM en utilisant OpenCL, ce qui rend le code exécutable sur tous les GPUs.

La comparaison a donc impliqué d'essayer de faire des exemples similaires afin de pouvoir comparer au mieux les deux implémentations des méthodes.

La stabilité numérique des deux méthodes SPH et LBM repose principalement sur l'initialisation de celles-ci, c'est pourquoi la recherche des paramètres rendant les deux méthodes identiques est un problème très complexe.

## 7.1 Initialisation de la simulation

La condition initiale joue un rôle crucial dans la comparaison des deux codes car l'objectif est de comparer les deux codes grâce à la modélisation du même phénomène physique. La première idée était de modéliser une goutte, en forme de cube car plus facile à définir, tombant dans un bassin d'eau, comme il est possible de voir sur la figure 7.1.

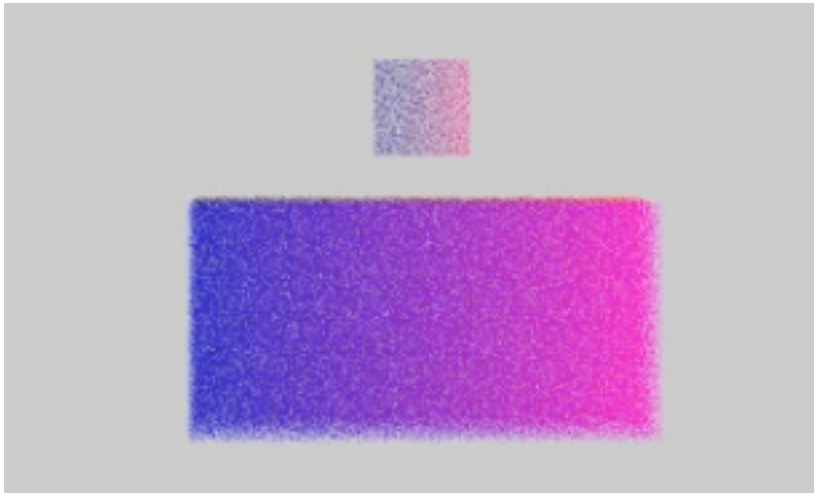
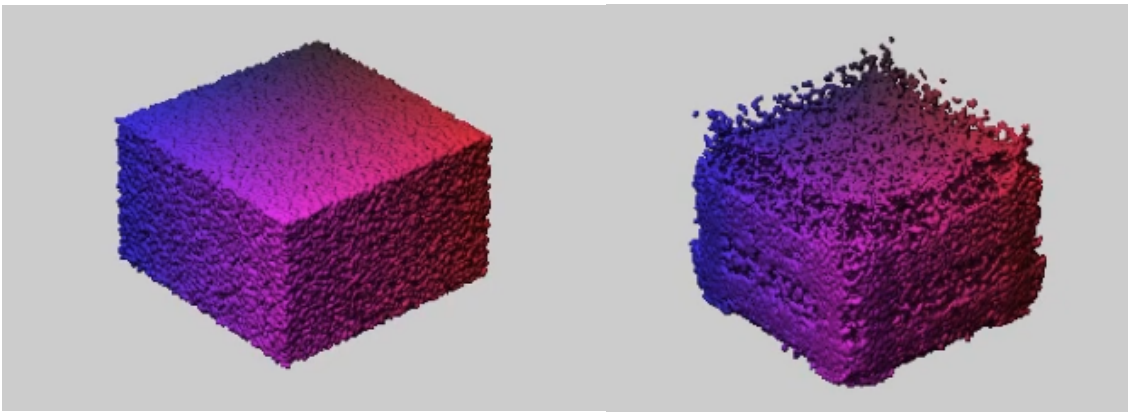


Figure 7.1: Idée initiale

À cet exemple, le travail a montré que la définition d'un bac d'eau était presque impossible dans l'implémentation de la méthode SPH, car celle-ci possède des instabilités au niveau des bords du domaine, c'est pourquoi le fluide implosait ou s'éparpillait en volant dans tout le domaine, comme il est

possible de voir sur la figure 7.2. En plus du phénomène d'implosion, le code s'arrêtait au bout d'un petit moment à cause d'accès illégaux à la mémoire.



(a) Condition initiale (b) Implosion

Figure 7.2: Difficulté de définir un bac d'eau

Face à ces difficultés, la solution finale a été de renoncer à cette comparaison qui aurait été idéale car elle aurait permis de faire varier la quantité d'eau dans le bac, permettant de voir à quel point le code avec domaine adaptatif était avantagé comparé à l'autre code qui aurait eu plus de peine à tourner pour un bac très rempli d'eau car il aurait dû calculer ce qui se passe au fond du bac, ce qui n'aurait pas été le cas du code avec domaine adaptatif.

C'est pourquoi l'exemple choisi est un exemple fonctionnel modifié de la méthode SPH.

Il s'agit d'un cuboïd d'eau tombant dans un bac, comme il est possible de voir sur la séquence d'images **?@fig-sequence** présente en annexe. Cet exemple a été reproduit dans les deux codes, c'est pourquoi la séquence d'images **?@fig-sequence** est tirée de FluidX3D. Dans l'exemple original, il y avait ensuite une force fréquentielle qui créait des vagues dans le bac, mais celle-ci a été supprimée afin de pouvoir avoir le même exemple dans les deux codes.

Comme le code GVDB est conçu pour modéliser des simulations à grande échelle comme l'océan par exemple et que le code de FluidX3D est plus dans le domaine du mètre, les deux exemples ne sont théoriquement pas identiques, c'est pourquoi les paramètres ont été définis afin que les deux codes donnent visuellement les mêmes résultats.

### 7.1.1 Moments clés

Lors de la simulation, 2 moments clés ont été défini. Le premier moment clé est utilisé afin d'ajuster la viscosité pour avoir un résultat similaire et le deuxième moment clé indique la fin de la simulation, comme il est possible de voir sur la figure 7.3.

Sur la figure 7.3, il est possible de remarquer que le volume d'eau n'est pas le même dans le code de GVDB que dans le code de FluidX3D. En effet, le fluide s'écrase et se compresse systématiquement sur GVDB-Voxel, comme montré sur la figure 7.4.

Afin de résoudre le phénomène de compression, un jeu sur de nombreux paramètres du code de GVDB-Voxel, tel que sur la taille des particules, l'espacement, la densité, etc, a été effectué, mais sans succès.

C'est pourquoi les deux exemples ne sont pas identiques, mais cependant très similaires.

## 7.2 Exécution des codes

Dans cette partie, l'objet d'étude sera l'exécution des deux codes.

Les deux codes donnent une visualisation en temps réelle, c'est à dire que le rendu est fait en même temps que le calcul de la simulation.

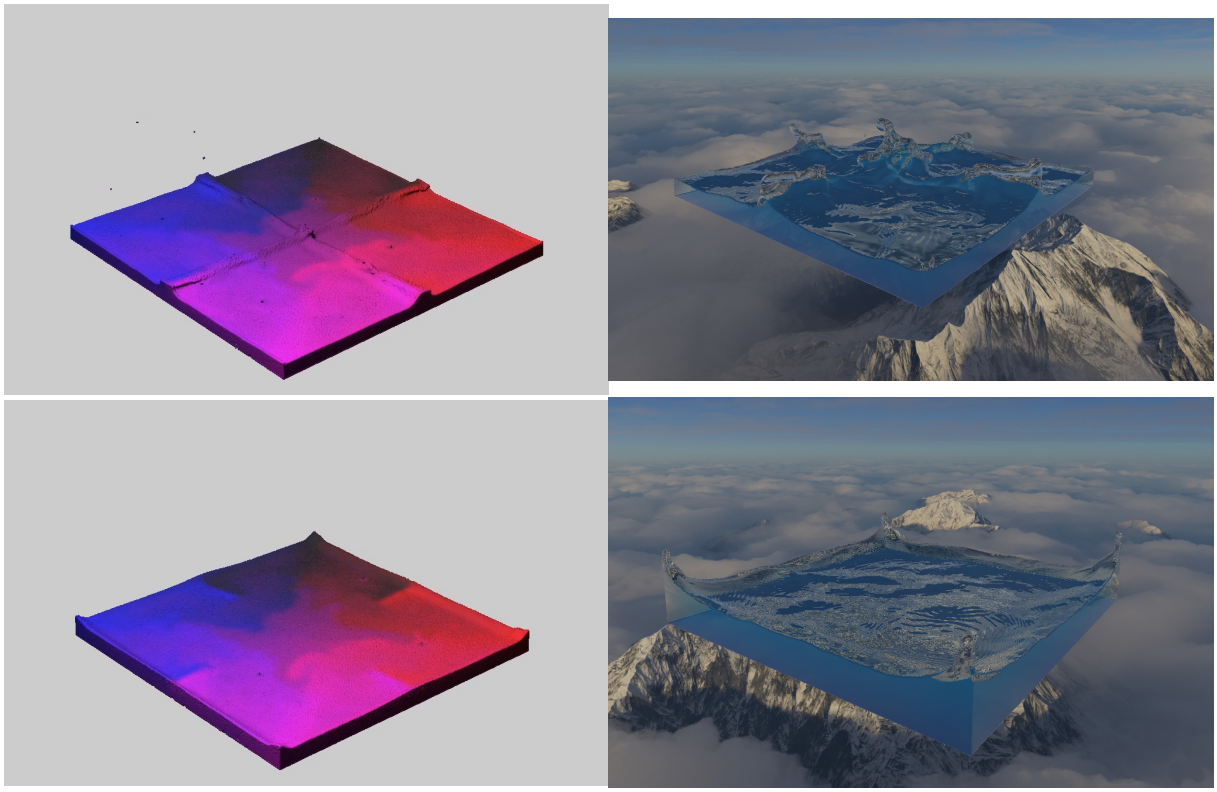
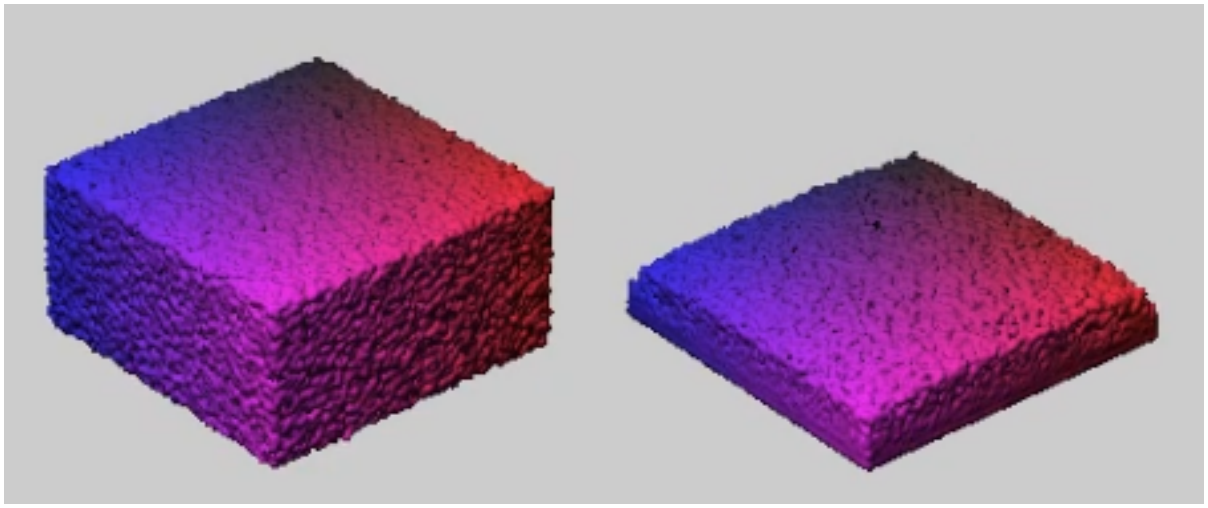


Figure 7.3: Moments clés



(a) Initialisation (b) Compression

Figure 7.4: Phénomène de compression dans GVDB

Afin d'évaluer les performances des deux codes, un script bash a été créé afin de reporter l'utilisation du CPU, de la RAM, et du GPU dans un fichier `csv`.

À cet instant du travail, un soucis s'est présenté car le code de FluidX3D ne fonctionne pas sur le linux de nom ordinateur car des erreurs XrandR sont apparues lors de l'exécution du code, erreurs qui sont restées irrésolues faute de temps et de documentation sur la provenance de ces erreurs. Mes suppositions à l'heure actuelle sont que Nvidia utilise X11 avec Xinemara qui est une autre version de XrandR, rendant l'affichage incompatible avec mon ordinateur linux. La solution n'étant pas trouvée, l'enregistrement des performances s'est fait sous Windows pour le code de FluidX3D.

Cela ne devait pas être trop problématique car les données principales qui nous intéressent, à savoir l'utilisation du GPU, se prennent avec `nvidia-smi` sur linux et Windows. Cependant, il a fallut traduire le script bash en script powershell, ce qui s'est fait avec l'aide de ChatGPT et des discussions sur les forums et les blogs.

Enfin, la visualisation des données a été faite à l'aide de python et de ChatGPT qui a donné le squelette du code, permettant d'économiser du temps de codage au profil du rapport.

Les résultats obtenus sont alors donnés sur les figures **?@fig-result-FluidX3D** et **?@fig-result-GVDB**.

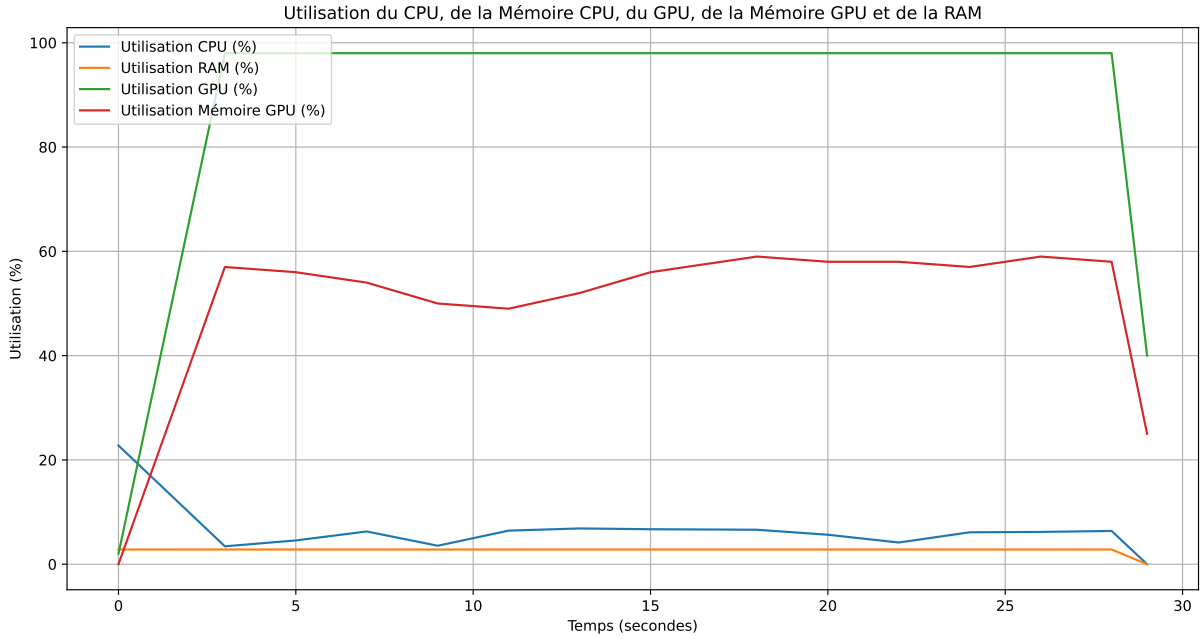



Figure 7.5: Résultats de FluidX3D

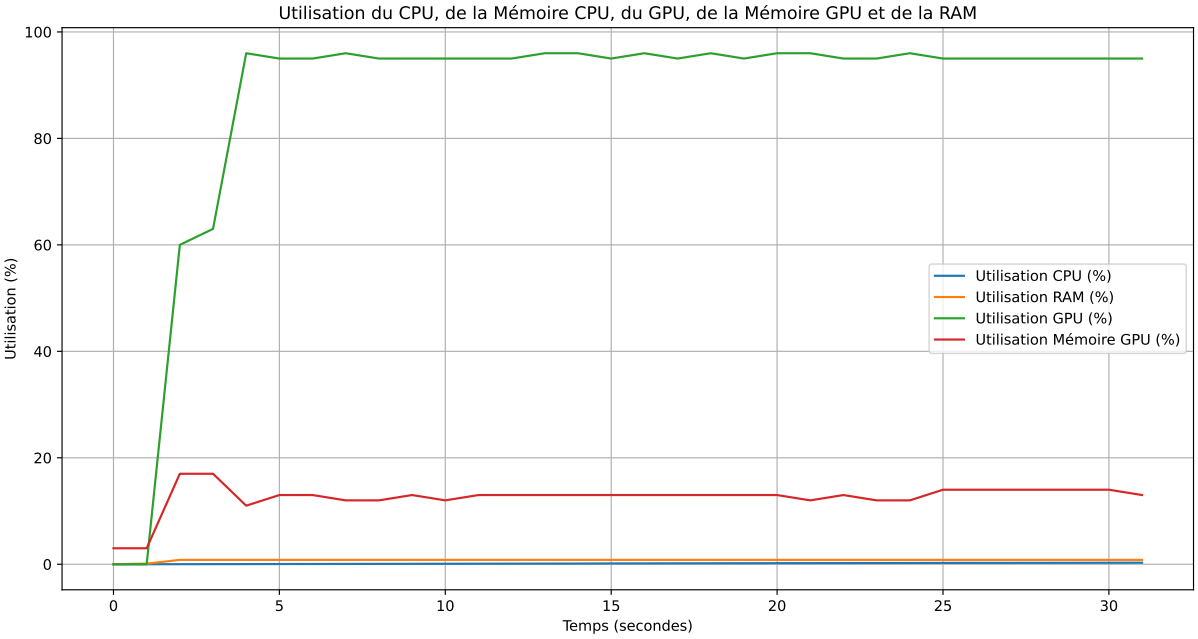



Figure 7.6: Résultats de GVDB Voxel

# Bibliographie


[1] S. Adami, X. Y. Hu et N. A. Adams. "A generalized wall boundary condition for smoothed particle hydrodynamics". In : *Journal of Computational Physics* 231.21 (2012), p. 7057-7075. DOI : 10.1016/j.jcp.2012.05.005.

[2] P.L. Bhatnagar, E.P. Gross et M. Krook. "A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems". In : *Physical Review* 94 (1954), p. 511-525.

[3] Fabien Caleyron. "Simulation numérique par la méthode SPH de fuites de fluide consécutives à la déchirure d'un réservoir sous impact". Français. NNT : 2011ISAL0103. tel-00711040. Autre. INSA de Lyon, 2011. URL : https://theses.hal.science/tel-00711040.

[4] S. Chen et al. "Lattice Boltzmann model for simulation of magnetohydrodynamics". In : *Physical Review Letters* 67 (1991), p. 3776-3780.

[5] NVIDIA Corporation. *GVDB Programming Guide*. Version 1.1. 2018. URL : https://developer.nvidia.com/gvdb.

[6] David Enskog. "Kinetische Theorie der Vorgänge in mässig verdünnten Gasen". Thèse de doct. Universität Uppsala, 1917.

[7] U. Frisch, B. Hasslacher et Y. Pomeau. "Lattice-gas automata for the Navier-Stokes equation". In : *Physical Review Letters* 56.14 (1986), p. 1505-1508. DOI : 10.1103/PhysRevLett.56.1505.

[8] R.A. Gingold et J.J. Monaghan. "Smoothed Particle Hydrodynamics: theory and application to non-spherical stars". In : *Monthly Notices of the Royal Astronomical Society* 181 (1977), p. 375-389.

[9] J. Hardy, Y. Pomeau et O. de Pazzis. "Time evolution of a two-dimensional classical lattice system". In : *Physical Review Letters* 31 (1973), p. 276-279.

[10] F. Higuera et J. Jimenez. "Boltzmann approach to lattice gas simulations". In : *Europhysics Letters* 9 (1989), p. 663-668.

[11] Jlcercos. *Figure*. https://commons.wikimedia.org/w/index.php?curid=70225405. Travail personnel, CC BY-SA 4.0. 2018.

[12] J.M.V.A. Koelman. "A simple lattice Boltzmann scheme for Navier–Stokes fluid flow". In : *Europhysics Letters* 15 (1991), p. 603-607.

[13] Carolin Körner et al. "Lattice Boltzmann Model for Free Surface Flow for Modeling Foaming". In : *Journal of Statistical Physics* 121 (oct. 2005), p. 179-196. DOI : 10.1007/s10955-005-8879-8.

[14] Kostorz. "A semi-analytical boundary integral method for radial functions with application to Smoothed Particle Hydrodynamics". In : *Journal of Computational Physics* 417 (2020). DOI : 10.1016/j.jcp.2020.109565.

[15] Timm Krüger. *Introduction to Lattice Boltzmann Method*. https://www.youtube.com/watch?v=jfk4feD7rFQ. ESPResSo Simulation Package. 2021.

[16] Moritz Lehmann. "High Performance Free Surface LBM on GPUs". Biofluid Simulation and Modeling, Theoretische Physik VI, Fakultät für Mathematik, Physik und Informatik. Masterarbeit. Universität Bayreuth, déc. 2019.

[17] Moritz Lehmann et al. "Accuracy and performance of the lattice Boltzmann method with 64-bit, 32-bit, and customized 16-bit number formats". In : *Physical Review E* 106 (juill. 2022). DOI : 10.1103/PhysRevE.106.015308.

[18] L.B. Lucy. "A numerical approach to the testing of the fission hypothesis". In : *Astronomical Journal* 82 (1977), p. 1013-1024.

[19] G. McNamara et G. Zanetti. "Use of a Boltzmann equation to simulate lattice-gas automata". In : *Physical Review Letters* 61 (1988), p. 2332.

[20] Liam Mooney. *Introduction to Containers and Docker*. https://endjin.com/blog/2022/01/introduction-to-containers-and-docker. Apprentice Engineer III. Jan. 2022.

[21] Ulf Schiller, Timm Krüger et Oliver Henrich. “Mesoscopic Modelling and Simulation of Soft Matter”. In : *Soft Matter* 14 (nov. 2017). doi : 10.1039/C7SM01711A.

[22] M. S. Shadloo, G. Oger et D. Le Touzé. “Smoothed particle hydrodynamics method for fluid flows, towards industrial applications: Motivations, current state, and challenges”. In : *Computers and Fluids* 136 (2016), p. 11-34. doi : 10.1016/j.compfluid.2016.05.029.

[23] Alban Vergnaud. “Améliorations de la précision et de la modélisation de la tension de surface au sein de la méthode SPH, et simulations de cas d’amerrissage d’urgence d’hélicoptères”. Français. ffNNT: 2020ECDN0033ff. fftel-03185147f. Mécanique des fluides [physics.class-ph]. École centrale de Nantes, 2020.

[24] Marcus Vesterlund. “Simulation and Rendering of a Viscous Fluid using Smoothed Particle Hydrodynamics”. Mém. de mast. Unknown, déc. 2004.