Annexe TIPE

Partie 1 : théorisation du TIPE

Eymeric CHAUCHAT

5 Juin 2021

Table des matières

1 Introduction

Le but de ce projet et de créer un simulateur de fluide inertiel pour prévoir les zones inondées à cause de la rupture d'un barrage.

La méthode utilisé pour réaliser ce simulateur est basé sur la méthode FLIP : la position et la vitesse du fluide est transporté par des particules (vision Lagrangienne) mais cependant la résolution de l'équation du mouvement se fait sur une grille (vision Eulerienne). Ainsi durant la résolution on passe successivement d'une grille a des particules.

Principat

Pour l'ensemble de l'étude on va prendre différentes notations :

L'espace des cases possède une taille E_m, E_n, E_p et les coordonnées des cases sont pris en leur centre . Les cases sont carrées et possèdent une largeur de ϵ_c

L'origine est $(0, 0, 0)$.

Tout les vecteurs sont dans $\mathbb{R}^3$.

Les Staggared Grid possèdent $(E_m + 1, E_n + 1, E_p + 1)$ composantes tandis que les grilles normales possèdent (E_m, E_n, E_p) composantes.

2 Outils pour la simulation

D'un point vue purement pratique la simulation nécessite plusieurs outils mathématiques pour se dérouler

2.1 Type de grille

Pour coder la simulation, j'ai fait le choix d'utiliser deux types de grilles différentes afin d'optimiser les calculs différentiels : Le premier est une grille de 3 Dimensions fig.1 et la seconde est une grille "décalée" qui permet de simplifier les calculs de dérivées spatiales fig.2

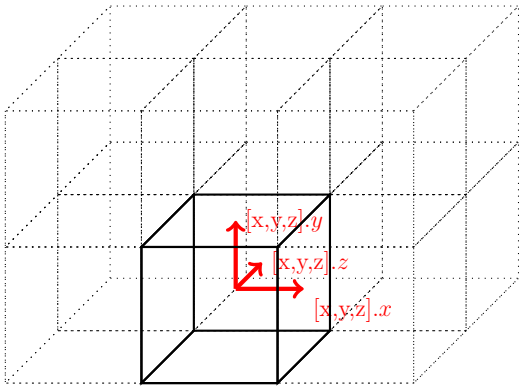

FIGURE 1 – Normal Grid

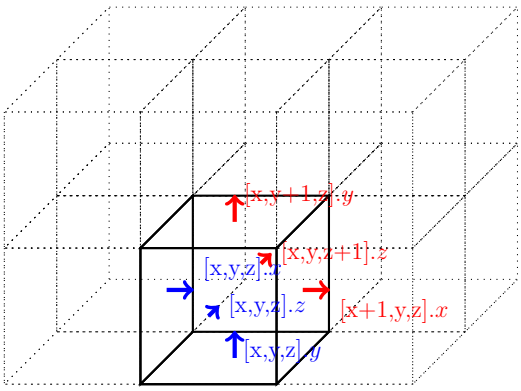

FIGURE 2 – Staggared Grid

2.2 Inversion de Jacobi

Algorithme

Pour inverser la matrice afin de trouver la solution de l'équation de Poisson, on utilise la méthode de Jacobi qui est une méthode d'inversion parallélisable itérative. On considère A une matrice à diagonale dominante

$$Ax = b$$

On décompose alors A suivant sa diagonale, sa matrice triangulaire supérieure et inférieure

$$A = D + L + U$$

On en déduit alors que :

$$\begin{aligned} \implies \quad & (D + L + U)x = b \\ \implies \quad & Dx = b - (L + U)x \\ \implies \quad & x = D^{-1}(b - (L + U)x) \end{aligned}$$

On peut alors définir une suite :

$$\forall n \in \mathbb{N}, x_{n+1} = D^{-1}(b - (L + U)x_n) \tag{1}$$

Cette suite converge car la matrice A est à diagonale dominante

Preuve de la convergence de la suite

Pour prouver la convergence de cette suite on utilise la norme infinie ($\|\cdot\|_\infty$) sur l'ensemble des matrices $M_n(\mathbb{R})$. Si on note $\forall n \in \mathbb{N}, v_n = x_n - x_{n-1}$ on a alors :

$$\begin{aligned} x_{n+1} - x_n &= D^{-1}(L + U)(x_{n-1} - x_n) \\ v_{n+1} &= D^{-1}(L + U)v_n \\ v_n &= (D^{-1}(L + U))^n v_1 \end{aligned}$$

Or sachant que l'on a $\|D^{-1}(L + U)\|_\infty < 1$ car la matrice A est à diagonale strictement supérieure on a par comparaison avec les suites géométriques que $\sum_{n=0}^{\infty} v_n$ Converge et ainsi on a que x_n converge.

2.3 Interpolation et dérivé discrète

2.3.1 Interpolation

On considère un tableau de vecteurs de 3 dimensions A représentant une grille décalée.

Pour interpoler les différentes valeurs de la grille aux particules on utilise une simple interpolation entre les coordonnées de la case de la grille avec la position des particules. On note la position de

la particule (p_x, p_y, p_z), qui transporte un vecteur (v_x, v_y, v_z) et la position de la case la plus proche (c_x, c_y, c_z) possédant un indice (m, n, p) (déterminé grâce à une division par la taille de la case)

On a donc trois coefficients de proportionnalités :

$$a_x = (p_x - c_x)/\epsilon_c$$
$$a_y = (p_y - c_y)/\epsilon_c$$
$$a_z = (p_z - c_z)/\epsilon_c$$

pour passer de la particule à la case on peut donc faire l'opération suivante :

$$A[m,n,p].x \mathrel{+}= a_x * v_x$$
$$A[m,n,p].y \mathrel{+}= a_y * v_y$$
$$A[m,n,p].z \mathrel{+}= a_z * v_z$$

$$A[m+1,n,p].x \mathrel{+}= (1 - a_x) * v_x$$
$$A[m,n+1,p].y \mathrel{+}= (1 - a_y) * v_y$$
$$A[m,n,p+1].z \mathrel{+}= (1 - a_z) * v_z$$

Si on a besoin de faire une moyenne des résultats (cas du transfert de vitesse etc...) on stocke les différents (a_x, a_y, a_z) dans un tableau de scalaires représentants les poids.

L'opération inverse se fait en inversant l'opération :

$$v_x = A[m,n,p].x * a_x + A[m+1,n,p].x * (1 - a_x)$$
$$v_y = A[m,n,p].y * a_y + A[m,n+1,p].y * (1 - a_y)$$
$$v_z = A[m,n,p].z * a_z + A[m,n,p+1].z * (1 - a_z)$$

2.3.2 Dérivée discrète

On dérive discrètement en se servant des Staggared Grid : On va avoir besoin du gradient d'un champ (en s'aidant de nos grilles le gradient de valeurs contenu dans une grille normale se retrouvera dans une Stag-Grid) :

On va prendre pour exemple le calcul du gradient du champ de pression $\vec{\nabla} p$:

Équation en coordonnées (ne pas oublier que $(x + \frac{1}{2}, y + \frac{1}{2}, z + \frac{1}{2})$ est le centre d'une case) :

$$\vec{\nabla} p_{x+\frac{1}{2},y+\frac{1}{2},z+\frac{1}{2}} \cdot \vec{u_z} = \frac{p_{x+\frac{1}{2},y+\frac{1}{2},z+\frac{1}{2}} - p_{x+\frac{1}{2},y+\frac{1}{2},z-\frac{1}{2}}}{\epsilon_c}$$

Équation de la matrice :

$$\overrightarrow{StgGridP}[X,Y,Z] = \begin{vmatrix} \frac{GridP[X,Y,Z] - GridP[X-1,Y,Z]}{\epsilon_c} \\ \frac{GridP[X,Y,Z] - GridP[X,Y-1,Z]}{\epsilon_c} \\ \frac{GridP[X,Y,Z] - GridP[X,Y,Z-1]}{\epsilon_c} \end{vmatrix}$$

On va aussi avoir besoin de la divergence d'un champ de manière discrète : $\nabla \cdot \vec{A}$

$$\vec{\nabla} \cdot \vec{A} = \frac{\partial \vec{A}}{\partial x} + \frac{\partial \vec{A}}{\partial y} + \frac{\partial \vec{A}}{\partial z}$$

Équation en coordonnées : (ou $\vec{A}_{x,y,z} = u_{x,y,z}\vec{u_x} + v_{x,y,z}\vec{u_y} + w_{x,y,z}\vec{u_z}$)

$$(\vec{\nabla} \cdot \vec{A})_{x+\frac{1}{2},y+\frac{1}{2},z+\frac{1}{2}} = \frac{u_{x+1,y+\frac{1}{2},z+\frac{1}{2}} - u_{x,y+\frac{1}{2},z+\frac{1}{2}}}{\epsilon_c} + \frac{v_{x+\frac{1}{2},y+1,z+\frac{1}{2}} - v_{x+\frac{1}{2},y,z+\frac{1}{2}}}{\epsilon_c} + \frac{w_{x+\frac{1}{2},y+\frac{1}{2},z+1} - w_{x+\frac{1}{2},y+\frac{1}{2},z}}{\epsilon_c}$$

3 Équation des fluides

les équations de Navier Stokes sont

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla + \frac{1}{\rho}\nabla p = \vec{g} + \nu \nabla \cdot \nabla \vec{u} \tag{2}$$

$$\nabla \cdot \vec{u} = 0 \tag{3}$$

On va démontrer les équations en passant par une vision particulaire du fluide :
On cherche à appliquer le principe fondamental de la dynamique sur la particule de fluide

$$m\frac{D\vec{u}}{Dt} = \vec{F}$$

On a tout d'abord la force de gravité :

$$\vec{F_g} = m\vec{g}$$

Après on a la force de pression ; le fluide tend à se déplacer des endroits de hautes pressions vers les zones de basses pressions

$$\vec{F_p} = -V\nabla p$$

où V et le volume qu'occupe la sphère de fluide

Il y a enfin la force de viscosité qui est une force qui tend à ralentir la particule de fluide pour qu'elle obtienne la vitesse moyenne des vitesses de fluide environnantes. Or l'opérateur qui permet de calculer la différence d'une variable par rapport à sa valeur moyenne est le laplacien. On a donc la force de viscosité suivante :

$$\vec{F_v} = -V\mu\Delta u$$

ou μ est le coefficient de viscosité du fluide.

En combinant les équations et en remplaçant la masse et la vitesse par la densité du fluide $m = \rho V$ on obtient :

$$m\frac{D\vec{u}}{Dt} = m\vec{g} - V\nabla p - V\mu\Delta u$$

$$\frac{D\vec{u}}{Dt} = \vec{g} - \frac{1}{\rho}\nabla p - \frac{\mu}{\rho}\Delta u$$

4 Méthode de résolution

4.1 Hypothèse simplificatrice

Pour résoudre numériquement les équations de fluide on considère qu'il est non visqueux car on se place dans l'approximation d'un fluide inertiel. On va donc travailler avec les équations suivantes :

$$\frac{D\vec{u}}{Dt} = \vec{g} - \frac{1}{\rho}\nabla p \tag{4}$$

$$\nabla \cdot \vec{u} = 0 \tag{5}$$

4.2 Splitting

Pour la résolution on utilise le principe de Splitting : qui énonce que à l'ordre un, on peut séparer une même équation pour la résoudre en plusieurs étapes .

5 Solution Numérique

Notation utilisée pour la résolution

On va se munir de différents outils pour pouvoir résoudre le problème : on va avoir besoin d'une grille décalée de vecteurs pour stocker la vitesse au temps n $\overrightarrow{StgVel}_n[X,Y,Z] = (u^n_{x,y,z}, v^n_{x,y,z}, w^n_{x,y,z})$, d'une grille pour stocker la vitesse au temps $n+1$ $\overrightarrow{StgVel}_{n+1}[X,Y,Z] = (u^{n+1}_{x,y,z}, v^{n+1}_{x,y,z}, w^{n+1}_{x,y,z})$ et d'une grille de pression (scalaire) $Pression[X,Y,Z] = p_{x,y,z}$.

De plus on va avoir besoin d'une grille qui stocke les types des cases $Type[X,Y,Z]$ et une grille qui va stocker l'influence de chaque particule pour la vitesse moyenne d'une case $\overrightarrow{Weight}[X,Y,Z]$

5.1 Déroulement de l'algorithme

l'algorithme de résolution des fluides s'organise de la manière suivante :

1. On place les vitesses des particules dans la grille
2. On définit les types de cases
3. On résoud :
   a) On ajoute les forces extérieurs
   b) On met les conditions au limites
   c) Calcul de la pression
   d) Ajout de la pression
4. On Intègre temporellement
5. On applique les conditions au limites sur les particules

5.2 Principe de splitting

On va utiliser la méthode du splitting pour résoudre numériquement les équations.La méthode de Splitting revient à résoudre à l'ordre 1. On peut alors séparer l'intégration de la force de gravité et de la force de pression présente dans l'équation de Navier-Stokes (4).

Tout d'abord on intègre la force de gravité :

$$\overrightarrow{u_{t+dt}} = \vec{u_t} + \vec{g} * dt \tag{6}$$

ensuite on peut donc faire exactement pareil pour la pression. On cherche donc a approximer le champs scalaire de pression :

$$\overrightarrow{u_{t+dt}} = \vec{u_t} - \frac{1}{\rho}\nabla p * dt \tag{7}$$

Le champ de pression va être calculé de sorte à ce que le nouveau champ de vitesse satisfasse la condition d'incompréssibilité :

$$\nabla \cdot \overrightarrow{u_{t+dt}} = 0 \tag{8}$$

6 Détail des étapes de l'algorithme

6.1 Placement des particules dans la grille

On utilise les formules d'interpolations vues ci-dessus (2.3.2) :

On interpole la vitesse dans une Stag-Grid et les poids des particules dans une autres et après on normalise la valeur de ces vitesses.

6.2 Définition des types de classes

On a besoin de définir les différents types de cases : (on range ces types dans une grille normale)

— Tout d'abord toutes les cases sur la bordure de la simulation sont mises en tant que mur. Les cases qui donnent la forme de notre situation sont aussi mises en mur.

— Les cases qui possèdent une vitesse de la Stag-Grid non nulles, sont définies en tant que fluide.

— Les dernières cases sont laissées en tant que case d'air.

6.3 Résolution

6.3.1 Force extérieure

On ajoute les forces extérieures (force de gravité) grâce à une intégration d'euler (Sachant que la gravité ne dépend d'aucune variable dépendante du temps et de l'espace l'intégration d'euler est stable)

$$\overrightarrow{StgVel}_{n+1}[X,Y,Z].z = \overrightarrow{StgVel}_{n}[X,Y,Z].z - g * dt$$

6.3.2 Conditions aux limites

Le principal type de condition aux limites utilisé pour la simulation est une condition aux limites de Dirichlet sur les bords de la simulation : Si on considère $\vec{n}$ la normale au mur, on doit satisfaire la condition des vitesses suivantes :

$$\vec{v} \cdot \vec{n} = 0$$

Informatiquement, on doit mettre à 0 les composantes qui sont normales aux différentes cases murs :

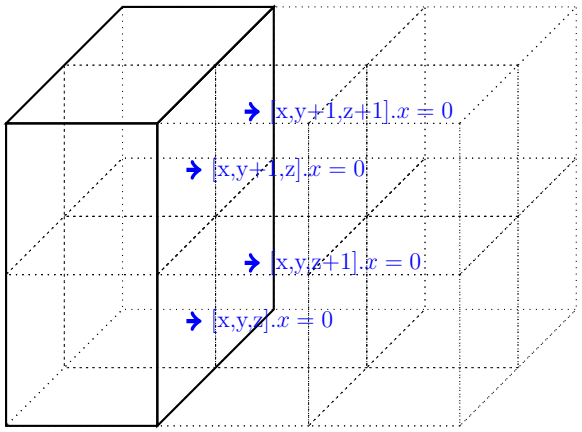

FIGURE 3 – Condition de Dirichlet

6.3.3 Détermination du champ de pression

Notre but va être de déterminer un champ de pression qui va vérifier les deux contraintes portées par les deux equations (8) et (7).

l'équation (7) implique avec les grilles et la dérivation discrète que :

$$\overrightarrow{StgVel}_{n+1}[X,Y,Z] = \overrightarrow{StgVel}_{n}[X,Y,Z] - \frac{1}{\rho}\nabla Pression[X,Y,Z] * dt$$

$$\begin{vmatrix} u^{n+1}_{x,y,z} \\ v^{n+1}_{x,y,z} \\ w^{n+1}_{x,y,z} \end{vmatrix} = \begin{vmatrix} u^{n}_{x,y,z} - \frac{dt}{\rho} * \frac{p_{x,y,z} - p_{x-1,y,z}}{\epsilon_c} \\ v^{n}_{x,y,z} - \frac{dt}{\rho} * \frac{p_{x,y,z} - p_{x,y-1,z}}{\epsilon_c} \\ w^{n}_{x,y,z} - \frac{dt}{\rho} * \frac{p_{x,y,z} - p_{x,y,z-1}}{\epsilon_c} \end{vmatrix} \quad (9)$$

On va expliciter la divergence discrète de (8) sous la forme :

$$\frac{u^{n+1}_{x+1,y,z} - u^{n+1}_{x,y,z}}{\epsilon_c} + \frac{v^{n+1}_{x,y+1,z} - v^{n+1}_{x,y,z}}{\epsilon_c} + \frac{w^{n+1}_{x,y,z+1} - w^{n+1}_{x,y,z}}{\epsilon_c} = 0$$

On réinjecte la première équation dans la seconde et on obtient :

$$\begin{aligned}
&\frac{u^{n+1}_{x+1,y,z} - u^{n+1}_{x,y,z}}{\epsilon_c} + \frac{v^{n+1}_{x,y+1,z} - v^{n+1}_{x,y,z}}{\epsilon_c} + \frac{w^{n+1}_{x,y,z+1} - w^{n+1}_{x,y,z}}{\epsilon_c} = 0 \\
&\frac{u^{n}_{x+1,y,z} - \frac{dt}{\rho} * \frac{p_{x+1,y,z} - p_{x,y,z}}{\epsilon_c} - u^{n}_{x,y,z} + \frac{dt}{\rho} * \frac{p_{x,y,z} - p_{x-1,y,z}}{\epsilon_c}}{\epsilon_c} + \\
&\frac{v^{n}_{x,y+1,z} - \frac{dt}{\rho} * \frac{p_{x,y+1,z} - p_{x,y,z}}{\epsilon_c} - v^{n}_{x,y,z} + \frac{dt}{\rho} * \frac{p_{x,y,z} - p_{x,y-1,z}}{\epsilon_c}}{\epsilon_c} + \\
&\frac{w^{n}_{x,y,z+1} - \frac{dt}{\rho} * \frac{p_{x,y,z+1} - p_{x,y,z}}{\epsilon_c} - w^{n}_{x,y,z} + \frac{dt}{\rho} * \frac{p_{x,y,z} - p_{x,y,z-1}}{\epsilon_c}}{\epsilon_c} \\
&\frac{u^{n}_{x+1,y,z} + v^{n}_{x,y+1,z} + w^{n}_{x,y,z+1} - u^{n}_{x,y,z+1} - v^{n}_{x,y,z+1} - w^{n}_{x,y,z+1}}{\epsilon_c} + \\
&(6p_{x,y,z} - p_{x+1,y,z} - p_{x-1,y,z} - p_{x,y+1,z} - p_{x,y-1,z} - p_{x,y,z+1} - p_{x,y,z-1})\frac{dt}{\epsilon_c^2 \rho}
\end{aligned} \quad \begin{aligned} &= 0 \\ &= 0 \\ &= 0 \end{aligned}$$

On obtient alors l'équation suivante :

$$\frac{u^{n}_{x+1,y,z} + v^{n}_{x,y+1,z} + w^{n}_{x,y,z+1} - u^{n}_{x,y,z} - v^{n}_{x,y,z} - w^{n}_{x,y,z}}{\epsilon_c} = (-6p_{x,y,z} + p_{x+1,y,z} + p_{x-1,y,z} + p_{x,y+1,z} + p_{x,y-1,z} + p_{x,y,z+1} + p_{x,y,z-1})\frac{dt}{\epsilon_c^2 \rho} \quad (10)$$

On peut désormais mettre cette équation dans la forme d'un système linéaire $Ax = b$ ou $x = (p_{0,0,0}, ..., p_{E_m,E_n,E_p})$ et $b = (u^n_{1,0,0} + v^n_{0,1,0} + w^n_{0,0,1} - u^n_{0,0,1} - v^n_{0,0,1} - w^n_{0,0,1}, ...)$: On peut expliciter la matrice A :

$$A = \begin{array}{c} \\ (En) \\ \\ \\ \\ \\ \\ (EnEm) \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cccccccccc} & & & & (En) & & & (EnEm) & & \end{array} \\ \begin{pmatrix} -6 & 1 & 0 & \cdots & 1 & 0 & \cdots & 1 & 0 & \cdots \\ 1 & -6 & 1 & \cdots & 0 & 1 & \cdots & 0 & 1 & \cdots \\ 0 & 1 & -6 & \ddots & & & \ddots & & & \ddots \\ \vdots & \vdots & \ddots & \ddots & & & & & & \\ 1 & 0 & & & & & & & & \\ 0 & 1 & & & & & & & & \\ \vdots & \vdots & \ddots & & & & & & & \\ 1 & 0 & & & & & & & & \\ 0 & 1 & & & & & & & & \\ \vdots & \vdots & \ddots & & & & & & & \end{pmatrix} \end{array} \frac{dt}{\rho \epsilon_c} \quad (11)$$

Pour trouver la matrice de pression, il faut désormais résoudre cette équation linéaire, c'est à ce moment là qu'on va utiliser l'algorithme de Jacobi pour pouvoir trouver les coefficients de p.

Généralisation de l'algorithme de résolution

Cette algorithme peut être étendue à l'intégration temporelle d'ordre supérieur (la démonstration présente dans ce document se base sur la méthode de splitting qui est d'ordre 1). Cependant la preuve aux ordres supérieur utilise le théorème de Helmotz-Hodge sur les champs vectoriels. On va donc admettre la preuve dans le cas général de n'importe quel ordre d'intégration .

Pour permettre une meilleure conservation du volume du fluide on va rajouter une force "d'explosion" qui va permettre de garder la bonne densité du fluide : pour se faire on va calculer la densité locale grâce aux nombres de particules dans une case et si la densité est plus élevée que la densité théorique du fluide alors au lieu d'imposer une divergence nulle on impose une divergence égale à notre manque de densité.

6.3.4 Soustraction de la matrice de pression

Lorsque l'on a la matrice de pression, il suffit d'appliquer l'équation (9) pour obtenir alors le champ de vitesses correspondant à celui d'un fluide en écoulement.

6.4 Intégration temporelle avec la méthode leapfrog

On utilise une intégration de leapfrog (aussi appelée en saute-mouton) pour faire avancer nos particules sur la grille : Ayant le champ des vitesses calculées en tous points de l'espace, on interpole la nouvelle vitesse d'une particule grâce à sa position, ensuite on calcule la vitesse à la position de la particule plus la vitesse interpolée fois un demi pas de temps et au final on utilise cette vitesse intermédiaire pour faire une intégration d'Euler (à noter que cette méthode se rapproche de la méthode de Runge-Kutta 2 mais on ne calcul pas le champs des vitesses a $t + h/2$ ce qui diminue la précision de la méthode par rapport à celle de Runge-Kutta 2).

7 Sous-script complément

Cette section présente les différents scripts utilisés qui permettent de faire entièrement fonctionner la simulation

7.1 Import sous Blender

On a un premier script qui permet d'importer les points de la simulation sur l'architecture de blender pour pouvoir faire le rendu final de la simulation.

7.2 Script des collisions

Pour pouvoir générer les collisions, on utilise un script qui prend en entrée un objet dans un fichier .stl et qui génère un fichier contenant la liste des murs :

Pour se faire, on utilise une succession de formules qui nous permettent de savoir si oui ou non un point est proche de la surface d'un triangle. On répète alors le procédé sur chaque triangle de notre objet pour obtenir la liste des cases qui seront des murs. Ainsi, il nous reste plus qu'à importer cette liste de points sur le buffer de la carte graphique pour pouvoir interpréter les cases à ces positions comme des murs (avec le même système de collision définit précédemment).