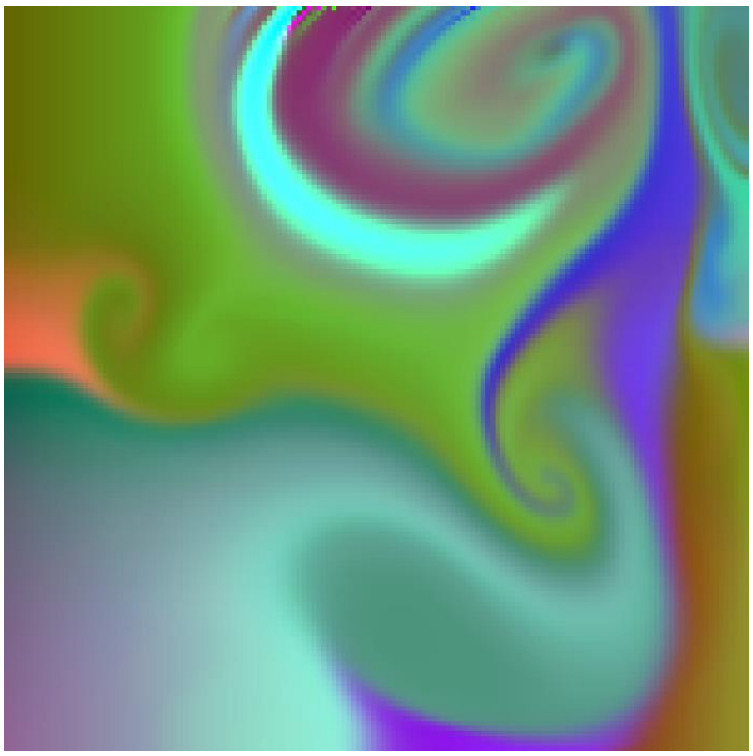
Projet – INF555

Simulation de fluides en 2D

A partir du papier *Stable Fluids* publié par Jos Stam

BESSA Mikhaïl
BARRAL Benjamin



Nous avons implémenté l'algorithme décrit dans *Stable fluids* pour simuler des mouvements de fluides (liquides incompressibles et gaz dans l'air), problématique importante en animation et effets spéciaux. Pour des raisons de temps et de puissance de calcul, nous avons mené une approche 2D. Nous allons présenter dans un premier temps l'article initial et l'algorithme, puis notre implémentation (nos interprétations, initiatives et différenciations), et enfin nos résultats.

I. Description de la méthode

L'article étudié présente une méthode de simulation de fluides, qui trouve un compromis entre une rigueur physique (provenant de la théorie de la dynamique des fluides) et des contraintes informatiques de vitesse d'exécution et de stabilité : le but est d'assurer un visuel réaliste avec des temps d'exécution raisonnables, et d'autoriser une interaction quelconque sans risque de divergence. L'algorithme consiste donc en une résolution numérique « expérimentale » des équations de Navier-Stokes, qui est plutôt rapide et inconditionnellement stable, ce qui a l'avantage d'autoriser des pas de temps relativement élevés.

Avant tout, le temps et l'espace sont discrétisés. Le pas de temps correspond au délai entre deux frames d'affichage graphique, et le fluide est divisé en cubes ou carrés, servant à la fois au rendu, et à la physique : pour certaines étapes on est amenés à concevoir les centres de pixels comme une particule microscopique.

Au niveau de la résolution, les trois points particuliers et innovants de la méthode de Jos Stam se résument de la façon suivante :

- Dans un premier temps, on transforme l'équation classique de Navier-Stokes par une manipulation mathématique qui consiste à projeter l'équation sur les vitesses dans l'espace des champs à divergence nulle (ce qui correspond physiquement à la condition d'incompressibilité), sur la base d'une décomposition de tout champ w en somme d'un champ à divergence nulle u et d'un gradient d'un potentiel q.
Cela a l'avantage d'éliminer le champ de pression de l'équation, et donne une intuition astucieuse pour faciliter la résolution (dernière étape dans le point suivant).

- L'algorithme de résolution consiste à considérer chacun des termes de l'équation séparément et à les résoudre de façon séquentielle, dans un ordre précis — qui n'est rigoureusement pas interchangeable, mais qui est intuitif et qui va du plus grossier au plus fin. En d'autres termes, à chaque frame, on résout successivement 4 étapes, où chacune consiste à composer le résultat de la précédente par une opération mathématico-physique :
  i. On additionne d'abord la contribution de la force : c'est logique de le faire en premier, car c'est ce terme qui déclenche le mouvement les particules.
  ii. Pour le terme d'advection : au lieu de résoudre ce dernier par un schéma explicite qui imposerait des conditions de stabilité, on adopte une approche « physique » qui consiste à advecter le champ d'une étape temporelle à l'autre, en retraçant les trajets des particules au moyen une méthode des caractéristiques, inconditionnellement stable.
  iii. Le terme de viscosité se résout comme une équation de diffusion par différences finies avec un schéma explicite (inconditionnellement stable).
  iv. Enfin la projection incompressible s'applique au résultat de la troisième étape : par différences finies on résout à la fois l'équation de Poisson pour le potentiel, et on calcule son gradient qui est retiré à la vitesse.

D'un point de vue informatique, l'aspect *laplacien* des matrices de résolution des étapes iii et iv rend la résolution inconditionnellement stable, et leur caractère creux offre une vitesse d'exécution convenable : un solutionneur linéaire permet en effet de résoudre ces systèmes de façon approchée, ce qui donne une complexité finale en O(N) (où N=n*n est le nombre de pixels considérés).

- Enfin, le rendu du mouvement se fait en advectant les textures à la manière de la résolution des vitesses : il s'agit de déplacer les pixels grâce à leurs vitesses.

II. Notre implémentation

1. Description

C'est la bibliothèque Processing qui est utilisée pour l'affichage de la simulation, et la bibliothèque MTJ pour la résolution des systèmes linéaires (notamment le *Conjugate Gradient Iterative Solver* pour les matrices creuses).

La fenêtre d'affichage (PApplet) est découpée en une grille de taille n*n pixels, chacun étant représenté par son centre dans les systèmes linéaires. Chaque centre est défini par ses coordonnées, sa vitesse, la force qui lui est appliquée, sa texture (couleur en RVB), et son scalaire (qui correspond à une concentration en particules). Toutes ces données sont contenues dans des tableaux de dimension 1 de taille n*n adaptés à la résolution de systèmes linéaires (les lignes sont concaténées). Les données vectorielles sont coupées en deux (composante verticale et composante horizontale).

L'animation du fluide repose sur la mise à jour de l'état des centres des pixels un certain nombre de fois par seconde (il faut plus de 15 images par seconde pour que la simulation soit fluide) à partir de la méthode décrite précédemment.

Nous avons implémenté deux modes de simulation :

i. **Mode « scalaire »** : l'utilisateur peut introduire des particules dans un fluide de couleur unie et y appliquer des forces. Les particules sont modélisées par un scalaire qui correspond à une concentration, et apparaissent dans une couleur plus ou moins opaque en fonction de cette concentration.
ii. **Mode « fluide visqueux » :** l'utilisateur peut simplement appliquer des forces à un fluide texturé.

De même, on a mis en place **deux types de conditions aux limites** : des conditions périodiques (ce qui sort d'un côté rentre du côté opposé) et des conditions « smooth plastic » i.e. où les vitesses normales sont nulles sur les bords (rien ne sort du cadre).

Avant de lancer la simulation, nous initialisons la grille de points (taille, couleurs initiales, nombre de points) et les matrices creuses servant à la résolution des systèmes. Ces dernières dépendent des conditions aux limites choisies (les bords des blocs composant les matrices changent). Il y en 3 pour l'étape de diffusion (une pour la vitesse horizontale, une pour la vitesse verticale et une pour le scalaire) et une pour l'étape de dissipation.

Notre projet est constitué de **quatre blocs principaux :**

i. **Gestion de l'animation et du temps réel** (initialisation, calcul des nouvelles images avec le solver, interactivité)
ii. ***Solver*** : nous en avons codé deux, chacun correspondant à un mode de simulation. Les deux héritent de la même classe abstraite. Ce bloc sert à mettre à jour l'état de chaque point entre deux images (donc à résoudre les systèmes à partir de la bibliothèque MTJ notamment).
iii. **Initialisation des matrices creuses** servant dans le solver en fonctions des paramètres, du mode et des conditions aux limites choisies.
iv. **Grille** : affichage des pixels correspondant aux points.

2. Interactivité

L'interactivité se repose sur des méthodes de Processing qui récupèrent des informations sur la souris/pavé tactile. En cliquant sur le fluide, l'utilisateur peut ainsi réaliser deux opérations :

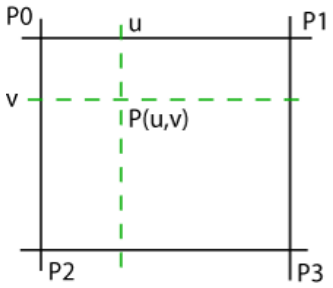
i. Le bouton gauche sert à appliquer des forces. En réalisant un cliqué-glissé, l'utilisateur définit l'intensité (vitesse et distance parcourue par le pointeur) et la direction (mouvement du pointeur) de la force qu'il va appliquer à l'ensemble des points survolés et leur voisinage. Deux modes sont disponibles : un ou les forces sont appliquées en temps réel avec le mouvement de la souris, et un autre on l'utilisateur donne des coups ponctuels.
ii. Le bouton droit sert à insérer des particules (terme source dans les équations) dans la zone autour du pointeur de la souris.

En outre, avec le clavier, l'utilisateur peut réinitialiser la simulation en changeant quelques paramètres (mode de simulation, conditions aux limites, paramètres, mode du bouton gauche…)

3. Routine de résolution

Voici quelques points de la « routine » de résolution qu'il faut préciser :

i. Fixer les forces/sources : le champ de force et les sources sont perpétuellement mis à jour en fonction des clics, et ceci indépendamment du calcul des frames. Avant le calcul d'une image, on fixe donc le champ de force et les sources.
ii. Transport des particules : pour suivre les particules, nous avons choisi d'appliquer un schéma de Runge-Kutta à l'ordre 1 sur toute la grille. C'est ensuite à partir d'une interpolation bilinéaire qu'on estime la vitesse et le scalaire en chaque point. De plus, nous avons pris le parti d'appliquer cette interpolation aux textures pour lisser la simulation (on l'applique en fait à chacune des coordonnées RVB). Cela entraine un mélange progressif des couleurs (synthèse additive) comme avec de la peinture (cf. première page). Il a fallu équilibrer cette interpolation de sorte à éviter de tendre progressivement vers le blanc ou au contraire vers le noir avec des erreurs d'arrondis accumulées.



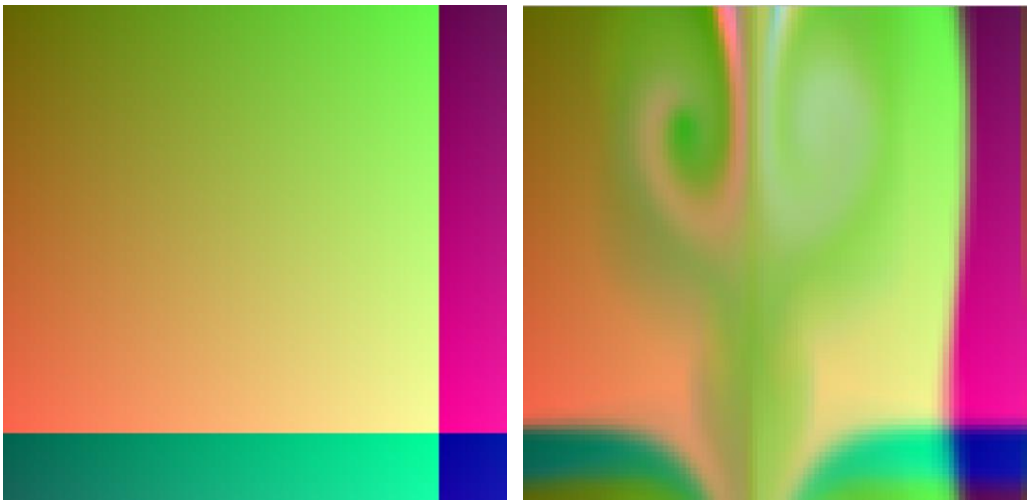
$$P(u,v) = ((1 - u) * P0 + u * P1) * (1 - v) + ((1 - u) * P2 + u * P3) * v$$

Interpolation bilinéaire d'une grandeur

Sur les bords, il faut adapter l'interpolation aux conditions choisies (périodiques ou pas).

III. Les résultats

La première satisfaction est que nous avons obtenu des mouvements qui sont visuellement réalistes, et que nous avons des résultats en temps réel pour des résolutions avoisinant les 100x100 mentionnés par l'article (**15 à 16 frames par seconde** pour cette résolution). Il faut cependant relativiser ces performances car l'article date de la fin des années 90.



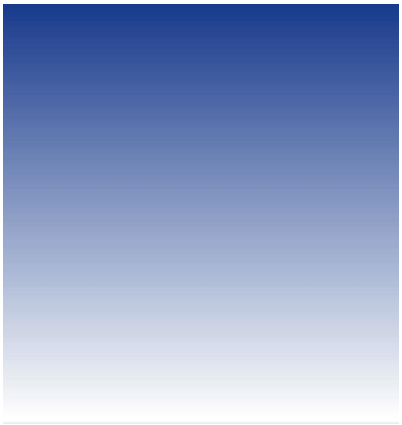
Un aperçu de l'apparence générale de nos fluides avant et après introduction de mouvement

Nous nous sommes avant tout concentrés sur la modélisation des **fluides visqueux**, mais avons également exploré la problématique de modélisation et l'**introduction de substances** (gaz, poussières…) introduites dans des fluides tels que de l'air.

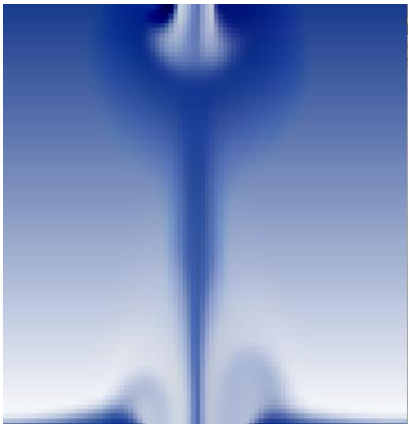
Nous avons ensuite cherché à évaluer le **réalisme** de nos résultats en concevant plusieurs **configurations** et en jouant sur différents **paramètres**. Les images suivantes donnent un aperçu de l'étendue des simulations possibles, et de la cohérence de celles-ci.

1. Les conditions aux limites :

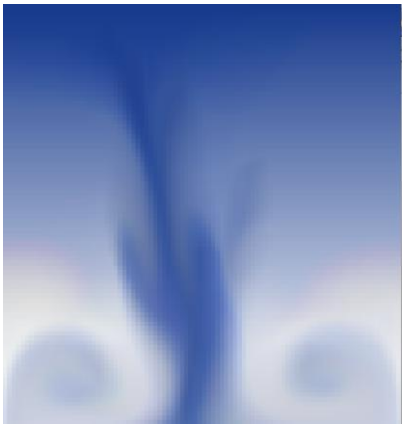
Les figures suivantes illustrent l'effet des deux conditions aux limites décrites précédemment. Pour une même force sensiblement verticale (mouvement de la souris de haut en bas) : voici les résultats obtenus.



Fluide au repos



Périodique



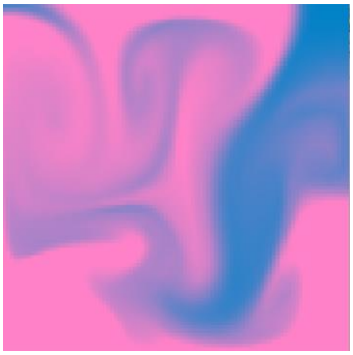
« Smooth plastic »

Dans le cas « périodique » les textures sortent de l'enceinte par le bas et entrent par le haut, tandis que dans le cas « *smooth plastic* » l'impénétrabilité repousse le fluide sur le côté au niveau de la paroi du bas.

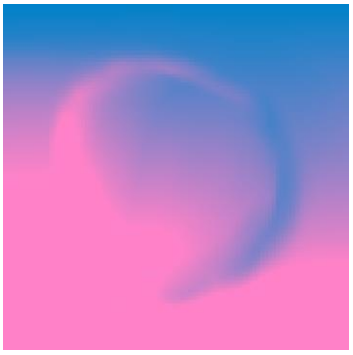
2. La viscosité

Pour évaluer la cohérence de nos résultats avec la valeur de la viscosité imposée, nous avons observé la réaction du fluide pour des viscosités différentes, en réponse aux mêmes forces (un trajet sensiblement « tourbillonnaire » de la souris ici).

Fluide au repos



Viscosité de l'eau



Viscosité intermédiaire

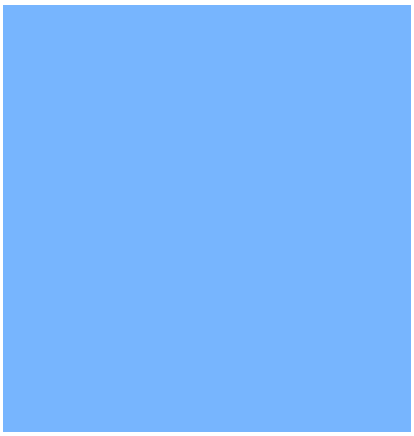
Viscosité de la glycérine

On constate dans le deuxième cas qu'une viscosité élevée a pour impact de freiner le fluide, jusqu'au cas « limite » de la glycérine (4ème figure) où les mouvements par à coup introduits par la souris sont freinés immédiatement sans se diffuser.

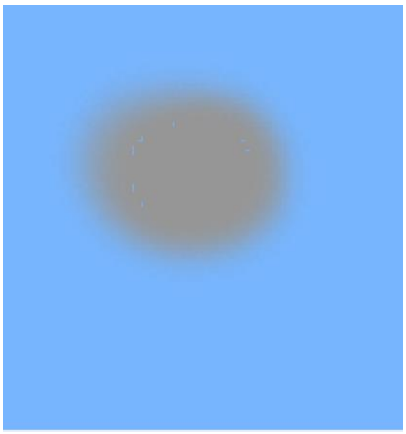
Remarque : les résultats ci-dessus sont à l'échelle de la fenêtre initiale (qui mesure à peu près 10 cm de largeur). Il a en effet fallu convertir les valeurs de viscosité en adéquation avec les unités de distance (en pixels graphiques) de Processing pour parvenir à des résultats cohérents.

3. L'introduction de substance

Voici ici un exemple de simulation d'introduction de substance. Nous avons modélisé un gaz qui s'apparente (par la couleur et le coefficient de diffusion) à du CO2 dans de l'air.



Avant introduction



Après introduction

Après mise en mouvement

La figure 2 montre la diffusion relativement lente, tandis que la figure 3 montre l'impact du mouvement du fluide (l'air) sur la répartition des particules.
Enfin, un constat intéressant sur l'ensemble des simulations est que, comme annoncé par l'article, l'**amortissement numérique** dû aux méthodes de résolution approchée se fait bel et bien ressentir, notamment et surtout proche des bords où les conditions aux limites rendent la discrétisation plus brutale. Cela est toutefois compensé par l'interaction qui permet de relancer le mouvement ou de réintroduire de la matière à tout instant.

Conclusion

Nous avons implémenté la méthode de Jos Stam pour simuler en temps réel et de façon stable les mouvements de fluides visqueux avec ou sans injection de particules, et une interface utilisateur permettant, par des actions de souris, d'imposer des forces ou d'injecter des particules au fluide.

Des pistes d'améliorations possibles de notre implémentation :

- Optimiser le code pour pouvoir supporter des résolutions toujours plus élevées
- Au niveau du traceur de particules, adopter un Range-Kutta d'ordre 2 dans les zones proches des bords où les différences de vitesse sont conséquentes et où un raffinement du pas de temps améliorerait le rendu de nos fluides.
- Au niveau richesse de simulation et d'expérience : modéliser des objets simples à l'intérieur du fluide (et donc sur le plan pratique concevoir de nouvelles matrices avec de nouvelles « conditions aux limites » internes) pour observer la réponse du fluide vis-à-vis d'un obstacle.
- Modéliser des forces plus réalistes
- Raffiner le modèle de rendu des particules (et la conversion densité-transparence).
- Le passage à la 3D.

Une application possible de notre programme serait de concevoir un jeu vidéo où le but serait d'envoyer un objet ou des particules dans des positions précises, par interaction avec le fluide.
Par ailleurs, avec une meilleure résolution, des textures initiales plus élaborées et des forces plus complexes, ce programme permettrait d'obtenir de très beaux rendus artistiques (vidéos et images).
Enfin, la méthode Jos Stam pourrait être adaptée pour modéliser des fluides irrotationnels : en remplaçant l'équation de Poisson par une équation de Laplace dans l'étape de « projection » de l'algorithme de résolution, on assurerait des écoulements à rotationnel nul.
Cette méthode expérimentale, plus généralement, pourrait être utilisée pour la résolution numérique de toute équation physique, pour lesquelles des conditions du type $div(V)=0$ sont imposées.