Méthodes simples pour l'animation de fluides en temps réel sur GPU

Martin Guay, Manuel Vennier

Méthodes simples pour l'animation de fluides en temps réel sur GPU

Martin Guay [1,2] et Manuel Vennier[1,2]

[1]INRIA Grenoble-Rhône-Alpes
[2]LJK (Université de Grenoble)

Résumé

Nous présentons des méthodes simples pour l'animation de fluides en temps réel sur GPU. Au coeur de nos méthodes se trouve une formulation dite « faiblement compressible »(weakly compressible) du fluide que l'on retrouve souvent du côté des Smooth Particles Hydrodynamics (WCSPH). Les méthodes basées sur des grilles sont plus adaptées aux structures de données manipulées par les GPUs, mais elles sont généralement basées sur une formulation implicite du champ de pression permettant de renforcer une condition de divergence nulle. Cette approche nécessite la résolution itérative d'un système de Poisson. Notre approche consiste à remplacer la formulation implicite de la pression par une formulation explicite (et donc simple) basée sur l'invariance de la densité du fluide. Cette formulation explicite nous permet de proposer un algorithme pour simuler des fluides monophasiques (fumée, feu) en une seule passe sur le GPU. Cette formulation de faible compressibilité basée sur la densité permet aussi, dans le cas de liquides, d'éviter plusieurs étapes liées au traitement d'une surface libre. Ainsi, nous proposons de l'utiliser dans une méthode hybride particules-grilles dans le style de FLIP (Fluid Implcit Particle) pour simuler des liquides sans avoir à extrapoler la vitesse au delà de la surface et à ré-initialiser un level set par exemple. De plus, nous pouvons traiter les particules indépendamment plutôt que d'avoir à reconstruire le voisinage des particules comme avec SPH. Nos méthodes sont très simples et donc rapides à implémenter sur GPU. Notre méthode en une passe pour les fluides monophasiques est moins précise et robuste que les méthode traditionnelles mais peut être utile a des fins éducatives, pour faciliter l'appréhension de la simulation de fluides. Enfin, dans le cas des liquides, notre méthode produit des résultats visuellement comparables à un algorithme WCSPH classique, et ce avec un gain en performance. Elle pourrait donc être utilisée dans des applications interactives comme des jeux vidéo.

We present simple methods to simulate fluids on the GPU in real-time. At the heart of our methods lies a weakly compressible formulation of fluids commonly found in Smooth Particles Hydrodynamics litterature (WCSPH), only here we use this formulation within a grid-based finite difference framework. On a GPU, grids are easier to manipulate than arbitrary particle configurations. However, Eulerian methods on the GPU typically involve iteratively solving a Poisson system for a pressure field representing the divergence part of the velocity field. We replace this implicit formulation of pressure by an explicit formulation based on density invariance (or weak compressibility). With this approach, we can propose a method to simulate 3D Eulerian gaseous fluids (single phase) that can solve for the motion of the fluid in a single pass on the GPU. Secondly, we propose a hybrid particle-grid method in the style of FLIP (Fluid Implicit Particle) to simulate liquids which allows treating the particles independently and neglecting the laborious tasks related to the free surface. Our results with the Eulerian single pass method are less accurate and robust than traditional methods strictly enforcing divergence-free. However, it can be useful for educational purposes such as an introduction to physics simulation or simply as a first working prototype where the pressure part can be replaced later on for accuracy and stability. Our simple liquid solver offers results comparable to an SPH solver and could therefore be used in a video game or other interactive application. We demonstrate the practicality of our method with examples of 2D and 3D gaseous fluids along 2D and 3D liquid-like phenomena.

Mots clé : Informatique Graphique, Animation de fluides, GPU

1. Introduction

L'emploi de modèles physiques pour l'animation de fluides est aujourd'hui le moyen le plus répandu pour

capturer le comportement chaotique des fluides ainsi que ses interactions complexes avec l'environnement. Mais bien qu'apportant une richesse visuelle importante, ces méthodes souffrent de critiques récurrentes sur leur caractère ésotérique et leur mise en oeuvre complexe et ce, particulièrement sur GPU. Comme en fait état [Hec11], une croyance répandue consiste à penser que la propriété prépondérante dans le choix d'une méthode pour l'industrie du jeu vidéo porte sur la rapidité d'exécution, alors que les premières qualités recherchées sont la robustesse et la simplicité. En effet, chaque nouvel algorithme intègre un écosystème contenant d'autres méthodes tournant en boucle et interagissant ensembles. Les possibilités de conflits et bogues sont alors très grandes, et leur identification et résolution complexes. Nous nous attaquons ici à cette complexité, en proposant des méthodes simples à mettre en oeuvre. Nous ne cherchons pas à être plus rapides ou visuellement plus attrayants, mais bien de pouvoir permettre, par de petites itérations de développement, l'obtention d'un solveur le plus simple possible. Un tel solver contiendrait déjà une étape d'advection et de visualisation que l'on retrouve aussi dans les méthodes classiques comme *Stable Fluids* [Sta99]. Ainsi il est possible de mettre en place un premier prototype fonctionnel et ensuite d'itérer vers une méthode robuste et précise.

Lorsque l'on cherche à simuler sur GPU des phénomènes gazeux comme la fumée ou le feu, l'approche la plus naturelle revient à employer une méthode Eulérienne sur grille. En effet la grille sur GPU semble beaucoup plus simple et naturelle que l'utilisation d'un ensemble de particules en mouvement. De plus, les méthodes Lagrangiennes sont peu adaptées pour la simulation de fluides monophasiques. Les particules n'étant pas influencées par la gravité, elles finissent par emplir tout le domaine et font ainsi perdre l'attrait des méthodes SPH qui normalement ces dernières permettent de concentrer les ressources sur la partie *visible*. En revanche, SPH est une méthode très employée en graphique pour les liquides puisqu'elle permet de contourner de coûteuses et délicates étapes associées au traitement de la surface libre, que l'on retrouve du côté des méthode Eulériennes. Il s'agit notamment de l'extrapolation du champ de vecteurs-vitesses au delà de la surface, et souvent, de la ré-initialisation d'une fonction distance (*level set method*).

Dans le but de proposer les méthodes les plus simples possible, nous proposons de rassembler les avantages de chacune de ces deux familles d'approches. Nous devons donc identifier les étapes les plus complexes, et tenter de les remplacer par des alternatives plus simples. Pour la simulation de gaz sur GPU, le travail introduit par Stam [Sta99] est le plus souvent employé. Les difficultés souvent perçues de cette méthode portent sur la détermination d'un champ de pression représentant la partie divergente du champ de vecteurs-vitesses. Pour se faire, un système de Poisson doit être résolu itérativement dans le but de renforcer une condition de divergence nulle. Nous remplaçons cette formulation implicite du champ de pression par une explicite, basée sur l'invariance de la densité, que l'on trouve souvent du côté de SPH. En utilisant cette formulation, nous arrivons à produire un solveur de fluides monophasiques pouvant résoudre les équations concernées en une seule passe, comme indiqué dans [GCE11]. Cette méthode étant limitée aux phénomènes monophasiques (feu, fumée), nous proposons donc une extension aux liquides.

Dans le cas des liquides, le traitement de l'interface air-eau exige plus de travail que dans le cas des fluides monophasiques. Pour leur simulation, il est souvent intéressant d'employer une formulation SPH qui, dans les cas simples, repose aussi sur une formulation d'invariance de la densité. Cette méthode permet de se concentrer sur la partie *visible*, c'est-à-dire le liquide, et d'omettre la représentation de l'air. Il est ainsi possible d'éviter les étapes associées à la surface libre que l'on rencontre du côté des méthodes Eulériennes décrites dans [EMF02]. La complexité des méthodes SPH sur GPU porte sur la reconstruction du voisinage de chaque particule. Pour échapper à cette étape laborieuse, nous formulons une méthode hybride similaire à FLIP (*FLuid Implicit Particle*) [ZB05] dans laquelle des particules représentent le liquide et plusieurs étapes de calcul s'effectuent sur une grille associée. Cette approche permet de traiter les particules indépendamment, d'effectuer un transport plus précis (*Runge Kutta* d'ordre deux par exemple) et de renforcer strictement une divergence nulle qui est une étape délicate avec des particules. Dans notre version, nous remplaçons la formulation de stricte incompressibilité par une formulation de *faible compressibilité*, mais l'employons ici sur grille dans un cadre de différences finies plutôt que de convolutions SPH. Ainsi, nous pouvons traiter les particules indépendament ainsi qu'éviter les laborieuses étapes liées au traitement de la surface libre. Nous obtenons des résultats similaires à un solveur WCSPH avec un gain en performance, et ce avec une méthode plus simple à implémenter sur GPU.

2. Travaux précédents

La simulation de fluides connaît un engouement manifeste dans la communauté graphique depuis plusieurs dizaines d'années déjà. Pionniers dans ce domaine, [KVH84] ont proposé une méthode procédurale pour simuler des nuages. Plus tard, [KM90] a employé une version linéarisée des équations de Saint-Venant pour obtenir des simulations en temps-réel, mais ce fut [FM96, FM97] qui le premier proposa une méthode d'animation de fluides permettant de résoudre en Eulérien les équations non-linéaires de Navier-Stokes. Leur méthode souffrait d'instabilités numériques dues à l'intégration explicite du terme de transport non-linéaire, et [Sta99] proposa l'emploi d'une méthode d'advection dite semi-Lagrangienne pour y remédier. Pour la fumée, [FSJ01] ont étendu l'approche de Stam avec une force chargée de confiner la vorticité, et ainsi préserver le plus possible les détails du fluide. En ce qui concerne la simulation de liquides, [EMF02] introduisit différentes étapes en Eulérien pour traiter la surface libre, en plus de proposer la méthode *Particle Level Set*, ayant pour but d'atténuer le problème de perte de masse du liquide. Utilisant un autre angle d'approche, [MCG03] popularise une méthode utilisant non plus une discrétisation sur grille, mais sur des particules d'un point de vu Lagrangien. Il proposa différents kernels pour résoudre les équations de

Navier-Stokes dans un cadre de convolutions SPH.

Le problème de perte de masse des méthodes *level set*, la perte d'énergie associée aux transports semi-Lagrangien ou encore le manque de précision des formulations *faiblement compressibles* de SPH amenèrent les chercheurs à combiner le meilleurs des différentes approches en des méthodes hybrides comme FLIP [ZB05]. Des particules sont employées pour effectuer le transport permettant de mieux préserver l'énergie, en conjonction d'une grille grossière pour les étapes de projection de divergence nulle. Cette amélioration de la précision des solutions est depuis au centre des préoccupations de nombreux travaux portant sur les fluides. Qu'il s'agisse de synthèse de turbulence sous-maille [SB08], de tracking explicite de surface et de modélisation de tension de surface [TWGT10], d'algorithmes de transport plus précis [SFB*07], ou de discrétisation pour l'interaction avec solides [BBB07], tous s'efforcent d'améliorer la précision. Nos travaux s'attaquant spécifiquement à la simulation de fluides sur GPU, nous présentons a présent un aperçu des travaux traitant cette problématique.

L'énorme puissance de calcul des GPUs en a fait des cibles de choix pour les simulations de fluides, très gourmandes en calculs. Le portage sur GPU de l'approche Eulérienne a été effectuée par [HBSL03], pour une simulation de nuages basée sur la méthode de [Sta99], méthode employée dans la majorité des solveurs de fluides monophasiques sur GPU. Pour ce qui est des liquides, [CLT07] employa une méthode de *level set*, mais son attrait est limité du fait d'une perte de masse importante et l'absence d'extrapolation de vitesse décrite dans [EMF02], rendant ainsi difficile le transport au delà de la surface. L'arrivée d'APIs dédiées au calcul (CUDA, OpenCL, DirectCompute) plus flexibles ont permit l'implémentation des étapes liées à la surface libre entraînant ainsi un regain de popularité pour la simulation de liquides en Eulérien. En 2011, [CM11] ont proposé une grille adaptative de *cellules colonnes* pour accélérer la résolution du système de Poisson. Un *level set* est employé pour représenter la surface et des détails sous-maillage tels que les éclaboussures sont ajoutées comme dans [CM10]. Le problème de la perte de masse a été récemment abordé par [CM12], en adaptant pour le GPU un schéma conservatif du transport. A notre connaissance, seul [Fla08] traite de la méthode hybride FLIP sur GPU, mais souffre des mêmes problèmes que [CLT07] en ce qui concerne la vitesse au delà de la surface du liquide.

Pour ce qui est des liquides, la méthode SPH à souvent été favorisée puisqu'elle permet d'éviter plusieurs étapes liées au traitement de la surface libre ainsi que pouvoir extraire une surface pour le rendu directement à partir des particules. Ainsi, beaucoup de solveurs de liquides sur GPU ont été proposés utilisant SPH. Le plus difficile dans une simulation SPH sur GPU est la recherche de particules voisines. Des textures peuvent être employées [HKK07, ZSP08] pour refléter l'emplacement des particules, mais par contrainte matérielle chaque texel ne peut contenir que quatre particules. Cela oblige à se servir d'une résolution de texture élevée, nécessitant ainsi une grande quantité de mémoire. Se rapprochant plus de notre méthode sur les liquides, [KC05] splattent dans une première passe les densités de chaque particule, puis traversent à nouveau les particules pour calculer et splatter le gradient de pression ainsi que les autres forces. Enfin, la grille peut-être échantillonnée et les forces appliquées aux particules. L'opération de splatting étant relativement coûteuse, nous ne splattons la densité et les vitesses qu'une seule fois et effectuons les calculs en différences finies sur la grille directement. L'augmentation de la flexibilité des GPUs ainsi que le développement d'APIs dédiées au calcul (CUDA, Direct Compute, openCL) a permis l'implémentation d'algorithmes sophistiqués de tri spatiaux comme dans [GSSP10]. Ces APIs étant encore restreintes à certains matériels, il est encore assez rare de trouver des fluides dans les jeux-vidéo car ceux-ci visent à fonctionner sur le plus de matériel possible. Enfin, notons l'emploi de la méthode SPH pour discrétiser et résoudre les équations de Saint-Venant sur GPU par [SBC*11].

3. Méthode

Notre approche est basée sur une formulation dite de *faible compressibilité* (*Weakly Compressible*), où le champ de pression est définit explicitement par rapport aux variations de densité par rapport à son état au repos. Ce champ de pression est ensuite utilisé pour modifier la vitesse du fluide de façon à transporter la densité à son état initial. On rencontre souvent cette formulation du côté de SPH dans un cadre de convolutions, mais nous l'employons ici à des fins de simplification sur une grille régulière en différences finies. Dans cet article, nous présentons en premier lieu les équations complètes de Navier-Stokes sous l'hypothèse de *faible compressibilité*. Nous pouvons ensuite discrétiser ces équations et formuler des méthodes permettant de simuler différents types de fluides. La première méthode permet de simuler des fluides monophasiques comme le feu et la fumée en une seule passe sur le GPU. Nous présentons ensuite une technique hybride particules-grille pour simuler des liquides.

3.1. Modèle Physique

À l'échelle macroscopique, le comportement d'un fluide peut être idéalisé par des principes simples de conservation de la masse ainsi que de conservation de mouvement :

$$\frac{\partial \rho}{\partial t} = -\mathbf{u}\cdot\nabla\rho - \rho\nabla\cdot\mathbf{u}, \quad (1)$$

$$\frac{\partial \mathbf{u}}{\partial t} = -\mathbf{u}\cdot\nabla\mathbf{u} - \frac{\nabla P}{\rho} + \nu\Delta\mathbf{u} + \mathbf{g} + \mathbf{f_{ext}}, \quad (2)$$

avec $\mathbf{u}$ la vitesse, ρ la densité, ν la viscosité, P la pression, $\mathbf{g}$ la gravité et $\mathbf{f_{ext}}$ les forces externes agissant sur le fluide (*e.g.* les objets et les interactions avec l'utilisateur). Ces équations doivent ensuite être complétées par des valeurs initiales ainsi que des conditions aux limites, comme celles de Neumann pour les obstacles dans le domaine, et des conditions du type surface libre pour les liquides. Il est important de noter que le terme de viscosité ν a souvent un rôle de « stabilisateur »lors de simulations employant des schémas d'intégration explicite comme c'est le cas lors de simulations SPH.

Pour résoudre ce système, il est répandu de séparer les composantes des équations puis de les résoudre une à une (voir [BM07]). Par exemple, les termes de transport peuvent être résolus avec une méthode d'advection semi-Lagrangienne et le terme de diffusion, en différences finies. Chaque étape peut ainsi bénéficier de méthodes de résolutions adaptées. La détermination du champ de pression à appliquer au fluide est une partie importante de la simulation, ainsi présentons-nous brièvement la méthode traditionnelle ainsi que notre approche alternative.

3.2. Invariance de la densité

Une hypothèse très largement employée dans les simulations Eulériennes porte sur l'incompressibilité du fluide à simuler. En effet, dans la plupart des situations, les fluides ne sont pas ou très peu compressibles. En employant cette hypothèse strictement, nous pouvons affirmer que la densité du fluide reste invariante dans l'espace et dans le temps ($\frac{\partial \rho}{\partial t} = 0, \nabla \rho = 0$), et l'équation (1) peut ainsi se ramener à $\nabla \cdot \mathbf{u} = 0$, *i.e.* une condition de divergence nulle. Cette approche est populaire en graphique car la divergence nulle est une condition à la formation de turbulence, visuellement attrayante.

Pour satisfaire cette contrainte, les méthodes traditionnelles font appel au théorème de décomposition d'Helmholtz-Hodge, afin de trouver un potentiel de pression dont le gradient annule la partie divergente du champ de vecteurs-vitesses (voir par exemple [FSJ01]). Ce potentiel de pression s'obtient alors par la résolution itérative d'un système de Poisson

$$\nabla \cdot \mathbf{u} = \nabla^2 P. \tag{3}$$

En relaxant l'hypothèse forte d'incompressibilité, on va permettre à la densité de varier selon l'équation (1). Le fluide résultant sera donc temporairement compressible. Le champ de pression est alors définit explicitement comme une fonction qui cherche à ramener la densité du fluide à sa configuration initiale :

$$P = k(\rho - \rho_0), \tag{4}$$

où k est une constante de raideur permettant de rendre le fluide plus ou moins compressible et ρ_0 est la densité initiale du fluide. Cette approche fut introduite à la communauté graphique par [DC96]. Nous utilisons la variante proposée par [BT07] qui permet une invariance de la densité plus stricte $P = k \max\left(\left(\frac{\rho}{\rho_0}\right)^\gamma - 1, 0\right)$ où γ est un entier impair allant généralement de 3 à 7. En employant cette formulation, nous pouvons évaluer explicitement la pression à partir de la densité ρ, et pouvons facilement calculer son gradient dans l'équation (2). Nous utilisons cette définition de la pression aussi bien dans notre méthode de fluides monophasiques que dans celle des liquides.

3.3. Fluides monophasiques

Afin de résoudre en une seule passe les équations (1) et (2) pour le cas des fluides monophasiques, nous utilisons des différences finies sur une grille uniforme centrée (valeurs placées au centre des cellules) ainsi qu'un transport semi-Lagrangien. Cette technique est particulièrement facile à implémenter sur GPU, et ce même avec les APIs 3D traditionnelles (*vertex* et *fragment shaders*). Pour un descriptif complet de la méthode, ainsi qu'un algorithme en pseudo-code GPU, nous invitons le lecteur à se référer à [GCE11]. Nous nous contentons ici de présenter l'algorithme dans ses grandes lignes.

En premier lieu, la densité du fluide est initialisée uniformément, c'est-à-dire que $\rho = \rho_0$ et $\nabla \rho = 0$. Nous affectons aux cellules contenant des obstacles la même densité ρ_0, ce qui permettra au cours de la simulation de réintroduire de la densité dans le domaine, afin de compenser pour la diffusion engendrée par les erreurs d'approximations numériques lors de l'intégration de l'équation de conservation de la masse. Ensuite, les termes de transport ($\mathbf{u} \cdot \nabla \rho$ et $\mathbf{u} \cdot \nabla \mathbf{u}$) sont résolus à l'aide d'une méthode semi-Lagrangienne (voir [Sta99]). En un mot, il s'agit de traiter chaque point $\mathbf{x}_{i,j,k}$ de la grille comme une particule fictive qui aurait été transportée par le champ de vecteurs-vitesses $\mathbf{u}^n$. On cherche donc à recopier la valeur de la fonction à l'emplacement précédent ($\mathbf{x}_{i,j,k} - \Delta t \mathbf{u}^n$) vers la position actuelle.

Une fois le solveur de base implémenté, il est possible d'améliorer les résultats de différentes façons. Bien que la méthode semi-Lagrangienne soit inconditionnellement stable, elle est peu précise ($O(\Delta x, \Delta t)$ en dessous de la condition CFL), surtout pour un terme non-linéaire comme le transport de vitesses. Afin de conserver davantage de détails de vorticité, très importants pour la richesse visuelle, il est préférable d'employer un schéma d'intégration d'ordre supérieur comme celui de *MacCormack* [SFB*07], de précision d'ordre deux ($O((\Delta x)^2, (\Delta t)^2)$). Ces schémas ne peuvent cependant généralement pas être implémentés en une seule passe, et ne suffisent pas à préserver toute la vorticité. Il est donc recommandé de renforcer la condition de divergence nulle autant que possible, en employant des itérations de Jacobi [CLT07] ou encore avec des méthodes multi-grilles [CM11]. Nous insistons sur le fait que notre méthode permet de mettre en place plusieurs des blocs nécessaires (advection, visualisation) à un solveur typique (*Stable Fluids*) sur GPU, permettant ainsi d'appréhender plus facilement la simulation de fluides. Notre méthode simple Eulérienne pour les fluides est limitée au fluides monophasiques, aussi présentons-nous maintenant une simple méthode pour les liquides.

3.4. Liquides

La plupart des méthodes employées pour simuler des liquides en Eulérien nécessitent des étapes de traitement supplémentaires au niveau de l'interface entre le liquide et l'air. Ces étapes, décrites dans [EMF02], consistent à extrapoler le champ de vecteurs-vitesses au delà de la surface, à représenter et transporter l'interface, (la méthode du *level set* nécessite une étape de ré-initialisation) et à imposer des conditions aux de bords de type surface libre lors de la résolution du système de Poisson. Du côté de

SPH, la différence de densité très importante entre l'air et l'eau permet de négliger l'effet de l'air, et il est possible de s'occuper uniquement du liquide. La formulation *faiblement compressible* fonctionne approximativement dans ce cas. Pour éviter la recherche de voisins employée en SPH, nous utilisons une grille voisine dans laquelle nous *splattons* toutes les fonctions comme dans FLIP [ZB05]. Plutôt que d'effectuer toutes les étapes liées à la surface libre en Eulérien, nous utilisons la formulation *faiblement compressible*, mais en différences finies sur la grille. Ainsi nous conservons cet avantage de SPH à ne pas devoir s'occuper de la surface libre, et de plus évitons d'avoir recours à la recherche de voisins, laborieuse sur GPU.

La première étape de notre méthode consiste à *splatter* la densité et la vitesse des particules sur une grille grossière. Une fois les fonctions transférées sur la grille, nous pouvons évaluer explicitement le champ de pression à partir de la densité et résoudre ensuite les termes restants de l'équation (2) en différences finies et intégration d'Euler explicite :

$$\mathbf{u}^{n+1} = \mathbf{u}^n + dt * \{-k\nabla(\rho + \rho_{Obstacle}) - \mathbf{g}\}.$$

Cette étape terminée, nous échantillonnons le champ de vecteurs-vitesses résultant sur la grille pour mettre à jour la vitesse des particules, produisant un solveur du type PIC (Particle In Cell). Une autre méthode consiste à passer seulement les différences entre les vitesses initiales (« splattées ») et les vitesses corrigées produisant ainsi un solveur du type FLIP. La première approche (PIC) introduit une forte diffusion, pouvant être employée comme un facteur de stabilisation ou comme paramètre de viscosité, tandis que la seconde conserve bien mieux l'énergie. Comme dans [ZB05], nous utilisons une combinaison convexe des deux, avec un ratio FLIP-PIC r :

$$\mathbf{u}_p = r * \mathbf{u}^{n+1}(\mathbf{x}_p) + (1-r) * (\mathbf{u}_p - \Delta\mathbf{u}),$$
$$\text{avec } \Delta\mathbf{u} = \mathbf{u}^n(\mathbf{x}_p) - \mathbf{u}^{n+1}(\mathbf{x}_p),$$

où $\mathbf{u}^{n+1}(\mathbf{x}_p)$ correspond à la vitesse échantillonnée sur la grille, et $\mathbf{u}_p$ à la vitesse portée par la particule. Pour la simulation de liquides, nous utilisons une valeur de r faible, autour de 0.05 (à 95% FLIP). Pour finir, l'emploi d'une grille permet d'appliquer plus facilement un algorithme de transport d'ordre deux (RK2 par exemple), en s'assurant que les particules ne quittent pas le domaine à chaque étape intermédiaire.

3.5. Equations de Saint-Venant

Il est possible de remarquer une similitude entre la formulation de notre méthode et les équations de Saint-Venant. En effet, notre formulation peut être vue comme une généralisation en 3D des équations 2D de Saint-Venant. Pour s'en rendre compte, il suffit de remplacer le gradient de pression de l'équation (2) par la formulation donnée en (4). On retrouve alors les équations de Saint-Venant dans le cas où la hauteur du sol h_{Sol} est constante, et que sont gradient est nul :

$$\frac{Dh_\rho}{Dt} = -h_\rho \nabla \cdot \mathbf{u}, \tag{5}$$

$$\frac{D\mathbf{u}}{Dt} = -g\nabla(h_\rho + h_{Sol}) + \mathbf{f_{ext}}, \tag{6}$$

où h_ρ est la hauteur du fluide, et h_{Sol} la hauteur du sol. En 3D, h_ρ devient ρ, et le champ de hauteur du sol h_{Sol} peut être interprété comme $\rho_{Obstacle}$ pour placer des obstacles 3D dans le domaine (voir section 3.6). Ainsi, en utilisant notre méthode en 2D, on peut simuler des vagues et interpréter la densité comme la hauteur de l'eau servant à déplacer un maillage représentant la surface et pouvant servir au rendu. Comme dans la plupart des méthodes SPH (voir entre autre [SBC*11]), nous ne traitons pas le terme de divergence dans l'équation (5) même s'il reste de la divergence dans le champ vecteurs-vitesses. L'avantage de cette manoeuvre est que la densité contenue par les particules ne se diffusera jamais.

3.6. Obstacles rigides

Le terme $\rho_{Obstacle}$ dans l'équation (2) modifiée est donc équivalent au terme 2D h_{Sol} dans l'équation (6). Ce terme permet de considérer des obstacles dans le domaine en définissant une fonction de distance dans ces objets. Nous évaluons le gradient de ces fonctions de distance sur la grille en différences finies, mais elles peuvent aussi être évaluées analytiquement dans le cas d'obstacles simples, ou encore d'utiliser des *Metaballs* ou autre fonctions implicites pour approximer des objets, permettant d'éviter de voxeliser des maillages, comme il doit être fait avec les méthodes Eulériennes traditionnelles (voir [CLT07]). Concernant les bords du domaine, nous imposons une condition de type Neumann. Dans notre cas, elle consiste à annuler (mettre à 0) la composante du vecteur-vitesse normale à l'obstacle (voir [GCE11]).

4. Détails d'implémentation

Le *splatting* est une opération coûteuse et il est utile de bien calibrer la taille des rayons ainsi que la dimension des grilles. Nous avons employé une heuristique qui consiste à utiliser environ une particule par cellule et de *splatter* sur environ un rayon de 3 cellules. Pour *splatter* la densité, nous définissons dans un *geometry shader* la distance de chaque *vertex* au centre de la particule et laissons le *rasterizer* effectuer l'interpolation entre les *vertices*. Dans le *fragment shader*, nous évaluons la densité comme dans [SP08]

$$\rho_i = m_i \sum_j W(\|\mathbf{x}_i - \mathbf{x}_j\|, l) \tag{7}$$

et la vitesse est normalisée par la convolution totale,

$$\mathbf{u}_i = \frac{\sum_j \mathbf{u}_j W(\|\mathbf{x}_i - \mathbf{x}_j\|)}{\sum_j W(\|\mathbf{x}_i - \mathbf{x}_j\|)},$$

en faisant attention de ne pas diviser par des valeurs trop près de 0. Nous employons le noyau de convolution suivant :

$$W(r,l) = \frac{e^{(1.0-r/l)} - 1.0}{e^{1.0} - 1.0}, \tag{8}$$

qui vaut 0 lorsque $r >= l$.

Feu et fumée : Pour faire de la fumée ou du feu il est nécessaire d'ajouter une simulation de *densité de particules* qui représente la densité de particules de fumée dans l'air

par example. Ensuite cette densité peut être transportée (advectée) par le champ de vecteurs-vitesse produit par notre méthode simple en une passe. Davantage d'explications peuvent être trouvées dans [GCE11] ou [CLT07].

5. Résultats

Nous présentons des exemples de nos méthodes simples pour simuler des fluides *faiblement compressibles*. Ces prototypes ont été testés sur plusieurs cartes graphiques, mais les performances des méthodes sont mesurées avec la même carte (Quadro 6000). Les grilles sont représentées par des textures de flotants 16-bits, précision que nous jugeons suffisante. Les particules sont « splattées »avec un rayon d'environ 3 cellules.

La figure (**??**) est une simulation Eulérienne 3D utilisant une grille de 64^3 et 100,000 particules passives. La simulation du fluide et le rendu des particules s'exécute à 560 FPS (*Frames Par Seconde*). Le transport de la vitesse est effectué avec du semi-Lagrangien et les particules en Euler explicite RK2. La figure (**??**) présente aussi une simulation Eulerienne en 2D avec une grille 256×256 pour la vitesse. Pour la fumée il faut ajouter une fonction de densité (grille 512×512 dans notre cas) et effectuer un transport passif sous le champ de vecteurs vitesse [FSJ01]. Nous avons utilisé du transport semi-Lagrangien, mais de meilleurs résultats seraient obtenus avec une méthode de *MacCormack* [SFB*07].

La figure (**??**) illustre un liquide 3D sous interaction avec utilisateur, obstacle de terrain et une pierre sphérique dans le domaine. La grille employée est de dimension 96^3 avec $125,000$ particules, $\Delta t = 0.01$, PIC-FLIP de 0.5 et une gravité de magnitude 4. Le prototype s'exécute à 80 FPS incluant la simulation, le rendu en raycasting avec 64 pas fixes d'intégration suivit d'un post process d'anti-aliasing FXAA sur un *frame buffer* de 1024×768. Dans la figure (**??**), on peut voir un example d'un jet de liquide synthétisé à l'aide de notre méthode et des mêmes paramètres de simulation.

La figure (**??**) représente une simulation d'eaux peu profondes en 2D (Saint-Venant). On y emploie $41,000$ particules sur une grille 256^2 et la simulation s'exécute à 320 FPS. Nous aurions aussi pu utiliser notre méthode en Eulerien ici. La densité du fluide est utilisée directement pour déplacer un maillage servant au rendu final. Il est possible d'interagir avec les vagues en ajoutant une force 2D sur tout le domaine ce qui est visible dans la vidéo.

Également dans la vidéo, on peut voir une comparaison 2D entre notre méthode et une méthode WCSPH traditionnelle avec algorithmes d'accélération de recherche de voisins. Nous avons pris l'implémentation dans le SDK de DirectX 11. Une image de cette comparaison se retrouve dans la figure (**??**). Dans notre implémentation, on retrouve $65,000$ particules sur une grille 256^2 avec un rayon de splatting de 3. Notre méthode est environ $1.6\times$ fois plus rapide que le WCSPH accéléré. Complètement à droite, on retrouve notre méthode avec des itérations de Jacobi en plus pour réduire la divergence. Nous n'extrapolons pas la vitesse au delà de la surface et on peut voir que les particules ont de la difficulté à s'échapper de la surface.

6. Discussion

Nous proposons des méthodes plus simples à appréhender et à mettre en oeuvre que les méthodes traditionnelles, mais pour simplifier nous avons dû relaxer l'hypothèse forte d'incompressibilité du fluide souvent employée en graphique (surtout en Eulérien) et avoir recours à une formulation moins précise. Cette formulation de *faible compressibilité* ne permet pas de renforcer une condition stricte de divergence nulle, ainsi il est plus difficile d'obtenir la formation de turbulence, phénomène visuellement très apprécié. Ceci dit, il est possible de réduire la divergence en effectuant en plus des itérations de Jacobi (ou autre) pour résoudre un système de Poisson. Cela permet de produire davantage de turbulence ainsi que de pouvoir augmenter le pas de temps. Cependant, sans extrapoler la vitesse au delà de la surface le fluide sera contraint à la surface et les particules auront du mal à s'en échapper. Ainsi le liquide se comportera davantage comme un fluide monophasique plutot qu'un liquide comportant des éclaboussures.

D'un autre côté, avec un solveur FLIP traditionnel [ZB05], les particules ont tendance à s'agréger et il faut constamment ré-échantillonner la position de ces dernières pour éviter qu'elles ne soient inutiles (cela pourrait expliquer le faible nombre de solveurs FLIP proposés sur GPU). Dans notre cas, les forces de pressions basées sur la densité des particules vont automatiquement repousser ces dernières et ainsi permet d'éviter qu'elles ne s'agrègent, préservant mieux le volume.

Toujours dans cet esprit de simplicité, nous nous servons de grilles centrées, connues pour causer des problèmes numériques [BM07]. Pour éviter ces problèmes, une solution communément citée dans la littérature consiste à utiliser des grilles dites *décalées* (*staggered grids*, [HW*65]). Mais le gain sensé être apporté ne nous a pas semblé suffisant comparé à leur complexité de mise en oeuvre, spécialement avec les APIs graphiques. Nous avons insisté à plusieurs reprises sur la possibilité d'implémenter nos méthodes sans avoir recours à des API de calcul GPU (CUDA, DirectCompute, OpenCL). Nous considérons cela comme un avantage, car bien que gagnant de plus en plus d'importance, ces APIs ne sont pas encore disponibles sur tous les matériels, et leur utilisation dans les jeux-vidéo encore marginale.

7. Conclusion

Ce travail comporte deux méthodes simples ayant comme point commun l'emploi sur grille de la formulation *faiblement compressible* du fluide. La première portant sur les fluides monophasiques, permet d'animer du feu et de la fumée en une seule passe, permettant ainsi de favoriser l'appréhension de la simulation physique des fluides sur GPU. L'autre méthode portant sur les liquides est proposée comme une alternative simple aux solveurs WCSPH traditionels sur GPU permettant d'éviter la recherche de voisins et ce, avec

un gain en performance. Nos méthodes simples pourraient aussi être utilisées dans le cadre d'un cours d'introduction à l'animation physique ou encore pour le développement rapide d'un prototype de fluides dans un jeu vidéo ou autre application interactive. Une fois un solveur de base implémenté, il est ensuite possible d'ajouter les méthodes classiques d'amélioration des fluides. On entend par là l'ajout de particules de vorticité pour stimuler la turbulence comme dans [SRF05], ou bien des modèles sous-maille comme [KTJG08] pour amplifier les détails de la simulation.

Références

[BBB07] BATTY C., BERTAILS F., BRIDSON R. : A fast variational framework for accurate solid-fluid coupling. In *Proceedings of ACM SIGGRAPH 2007* (2007).

[BM07] BRIDSON R., MÜLLER M. : Fluid simulation : Siggraph 2007 course notes, 2007. ACM SIGGRAPH 2007 courses.

[BT07] BECKER M., TESCHNER M. : Weakly compressible sph for free surface flows. In *Proceedings of Eurographics/ ACM SIGGRAPH Symposium on Computer Animation 2007* (2007), pp. 209–218.

[CLT07] CRANE K., LLAMAS I., TARIQ S. : *Real-time simulation and rendering of 3d fluids*, vol. 3 de *GPU Gems*. Addison Wesley, 2007, ch. 30.

[CM10] CHENTANEZ N., MÜLLER M. : Real-time simulation of large bodies of water with small scale details. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation 2010* (2010), pp. 197–206.

[CM11] CHENTANEZ N., MÜLLER M. : Real-time eulerian water simulation using a restricted tall cell grid. In *Proceedings of ACM SIGGRAPH 2011* (2011), no. 82.

[CM12] CHENTANEZ N., MÜLLER M. : Mass-conserving eulerian liquid simulation. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation 2012* (2012), pp. 197–206.

[DC96] DESBRUN M., CANI M. : Smoothed particles : A new paradigm for animating highly deformable bodies. In *Proceedings of Eurographics Workshop on Animation and Simulation* (1996), pp. 61–76.

[EMF02] ENRIGHT D., MARSCHNER S., FEDKIW R. : Animation and rendering of complex water surfaces. In *Proceedings of the 29th SIGGRAPH* (2002), pp. 736–744.

[Fla08] FLANNERY R. L. : *A Hybrid Fluid Simulation On The Graphics Processing Unit (GPU)*. Master's thesis, Texas A&M University, 2008.

[FM96] FOSTER N., METAXAS D. : Realistic animation of liquids. *Graphical Models and Image Processing. Vol. 58*, Num. 5 (1996), 471–483.

[FM97] FOSTER N., METAXAS D. : Modeling the motion of a hot, turbulent gas. In *Proceedings of ACM SIGGRAPH* (1997), pp. 181–188.

[FSJ01] FEDKIW R., STAM J., JENSEN H. W. : Visual simulation of smoke. *Proceedings of ACM SIGGRAPH* (2001), 15–22.

[GCE11] GUAY M., COLIN F., EGLI R. : *Simple and Fast Fluids*, vol. 2 de *GPU Pro*. A.K. Peters Ltd, 2011, ch. VII-3.

[GSSP10] GOSWAMI P., SCHLEGEL P., SOLENTHALER B., PAJAROLA R. : Interactive sph simulation and rendering on the gpu. In *Proceeding of Eurographics/ACM SIGGRAPH Symposium on Computer Animation 2010* (2010), pp. 55–64.

[HBSL03] HARRIS M., BAXTER W., SCHEUERMANN T., LASTRA A. : Simulation of cloud dynamics on graphics hardware. *Proceedings of the SIGGRAPH/Eurographics Workshop on Graphics Hardware* (2003).

[Hec11] HECKER C. : A game developer's wish list for researchers, 2011. Talk given at the 2011 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games.

[HKK07] HARADA T., KOSHIZUKA S., KAWAGUCHI Y. : Smoothed particle hydrodynamics on gpus. In *Proceedings of Computer Graphics International 2007* (2007), pp. 63–70.

[HW*65] HARLOW F., WELCH J., ET AL. : Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of fluids. Vol. 8*, Num. 12 (1965), 2182–2189.

[KC05] KOLB A., CUNTZ N. : Dynamic particle coupling for gpu-based fluid simulation. In *Proceedings of the 18th Symposium on Simulation Technique* (2005).

[KM90] KASS M., MILLER G. : Rapid, stable fluid dynamics for computer graphics. *Proceedings of ACM SIGGRAPH. Vol. 24*, Num. 4 (1990), 49–57.

[KTJG08] KIM T., THÜREY N., JAMES D., GROSS M. : Wavelet turbulence for fluid simulation. In *Proceedings of ACM SIGGRAPH 2008* (2008), no. 50.

[KVH84] KAJIYA J. T., VON HERZEN B. P. : Ray tracing volume densities. In *Proceedings of ACM SIGGRAPH* (1984), ACM, pp. 165–174.

[MCG03] MÜLLER M., CHARYPAR D., GROSS M. : Particle-based fluid simulation for interactive applications. In *Proceedings of Eurographics/ACM SIGGRAPH Symposium on Computer Animation 2003* (2003), pp. 154–159.

[SB08] SCHECHTER H., BRIDSON R. : Evolving subgrid turbulence for smoke animation. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation 2008* (2008), pp. 1–7.

[SBC*11] SOLENTHALER B., BUCHER P., CHENTANEZ N., MÜLLER M., GROSS M. : Sph based shallow water simulation. In *Virtual Reality Interactions and Physical Simulation (VRIPhys)* (2011), pp. 39–46.

[SFB*07] SELLE A., FEDKIW R., BYUNGMOON K., YINGJIE L., JAREK R. : An unconditionally stable maccormack method. *Journal of Scientific Computing. Vol. 35*, Num. 2-3 (2007), 350–371.

[SP08] SOLENTHALER B., PAJAROLA R. : Density contrast sph interfaces. In *Proceedings of Eurographics/ ACM SIGGRAPH Symposium on Computer Animation 2008* (2008), pp. 211–218.

[SRF05] SELLE A., RASMUSSEN N., FEDKIW R. : A vortex particle mathod for smoke, water and explositions. *ACM Transactions on Graphics (TOG). Vol. 24*, Num. 3 (2005), 910–914.

[Sta99] STAM J. : Stable fluids. In *Proceedings of the 26th ACM SIGGRAPH conference* (1999), pp. 121–128.

[TWGT10] THÜREY N., WOJTAN C., GROSS M., TURK G. : A multiscale approach to mesh-based surface tension flows. In *Proceedings of ACM SIGGRAPH 2010* (2010), no. 48, ACM.

[ZB05] ZHU Y., BRIDSON R. : Animating sand as a fluid. In *Proceedings of ACM SIGGRAPH 2005* (2005), pp. 965–972.

[ZSP08] ZHANG Y., SOLENTHALER B., PAJAROLA R. : Adaptive sampling and rendering of fluids on the gpu. In *Volume and Point-Based Graphics* (2008), Eurographics Association, pp. 137–146.