Algorithme de simulation de fluide.

Mise en garde

Nous travaillons dans une version Eulérienne de simulation de fluide, qui discrétise un milieu continu de molécules d'eau dans une grille. Dans notre situation nous travaillons avec une grille 3D, mais par raccourcissement des explications / formules, je décrirai le principe pour une grille 2D.

Principe fondamental

Formules générales et complexes de Navier-Stokes

La simulation de fluide (eau, air, fumées, etc…) suit la loi de Navier-Stokes décrite autour de 3 lois de conservation. La loi de conservation d'une variable est décrite ainsi :

$$\frac{\partial \phi}{\partial t} + \nabla \cdot (\phi \vec{V}) = S$$

Avec ϕ la variable de densité entrainée à la vitesse $\vec{V}$, et la production volumique S, en prenant en compte le temps t.

Les lois de Navier-Stokes applique une relation de conservation de :

- La masse (équation de continuité ou équation de bilan de la masse) :

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{V}) = 0$$

- La quantité de mouvement :

$$\frac{\partial (\rho \vec{V})}{\partial t} + \nabla \cdot (\rho \vec{V} \vec{V}) = \vec{\nabla} \cdot P + \rho \vec{g}$$

- La quantité d'énergie :

$$\frac{\partial (\rho E)}{\partial t} + \vec{\nabla} \cdot (\rho E \vec{V}) = \vec{\nabla} \cdot (P \cdot \vec{V}) + \rho \vec{g} \cdot \vec{V} + \nabla \cdot \vec{q} + \vec{\nabla} \cdot \vec{q}_R$$

Avec ρ la masse volumique du fluide (en kg/m^3), P le tenseur des contraintes (généralement la pression) en Pa, $\vec{g}$ la gravité et/ou toutes forces extérieures (en m/s^2), E l'énergie totale par unité de masse (en J/kg) et $\vec{q}$ et $\vec{q}_R$ le flux de chaleur dû à la conduction thermique et au rayonnement respectivement (en J/(m^2.s)).

Formule simplifiée de Stokes

Voilà pour la généralité. Par chance, M. Stokes a bien voulu simplifier ces formules dans le cas d'un fluide newtonien. Il suffit maintenant de satisfaire cette unique équation :

$$\rho \frac{\partial \vec{V}}{\partial t} + \rho (\vec{V} \cdot \overrightarrow{grad}) \vec{v} = \rho P - \overrightarrow{grad}\, p + \mu \Delta \vec{V} + \left(\mu' + \frac{\mu}{3}\right) \overrightarrow{grad}(div\, \vec{V})$$

Avec $\vec{V}$ la vitesse du fluide (en m/s), p la pression (en Pa) et μ et μ' la viscosité et viscosité seconde du fluide respectivement (en Pa.s).

On traite ici avec de l'eau de densité $\rho = 1$, qui est un fluide incompressible, donc $div\, \vec{V} = 0$. De ce fait, on peut réduire l'équation à :

$$\frac{\partial \vec{V}}{\partial t} + (\vec{V} \cdot \overrightarrow{grad}) \vec{v} = P - \overrightarrow{grad}\, p + \mu \Delta \vec{V}$$

Formules classiques pour les équations d'Euler

Dans la quasi-totalité des papiers concernant la simulation de fluide par la méthode Eulérienne, nous rencontrons cette équation de Stokes sous la forme :

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u + \frac{f}{\rho} - \frac{\nabla \mathrm{p}}{\rho}$$

Et la contrainte d'incompressibilité $\nabla \cdot u = 0$ avec $u = \vec{V}$ et f les forces externes ($f = P$ dans les équations précédentes).

Différence entre simulation Eulérienne et Lagrangienne

La simulation basée sur la méthode Lagrangienne est fondée sur l'utilisation de particules. L'algorithme émet un grand nombre de particules ayant les propriétés du fluide en question et simule leur réaction en prenant en compte l'environnement (les parois de la simulation) ainsi que les particules alentour. Les algorithmes classiques de ces simulations utilisent le principe de SPH (Smoothed-Particles Hydrodynamics) dans les algorithmes FLIP et PIC (FLuid-Implicit-Particle et Particle-In-Cell), et ont l'avantage de fonctionner dans un système de sans maillage (mesh-free) permettant notamment de simuler « facilement » les fluides même quand de gros obstacles se déplacent dedans (ex : un bateau sur l'eau, un avion qui se déplace, ou encore le fonctionnement de turbines). Néanmoins, la gestion des collisions (parois) et la notion d'incompressibilité des fluides sont les points faibles de cette méthode comparée à la méthode Euclidienne.

La méthode Euclidienne, elle, se base sur une grille. On représente chaque cellule comme contenant « du liquide », sans considérer les particules mais plutôt un continuum de matière. Par la diffusion et le déplacement « probabiliste » du continuum de chaque cellule vers les voisines, on simule la dynamique du fluide de manière plus discrétisée que par la méthode Lagrangienne. On parle aussi souvent de méthode semi-lagrangienne. Tout comme la qualité de la simulation Lagrangienne dépend du nombre de particules utilisée, celle de la méthode Euclidienne dépend de la résolution de la grille.

On va dans le projet de simulation sous-marine utiliser la méthode Euclidienne car on peut supposer l'environnement changer peu, et ainsi on peut tirer avantage du faible cout de calcul.

Données

- 2 matrices : vélocités $\vec{v}$ et vélocités précédentes $\overrightarrow{v_{prec}}$
- 1 flottant : viscosité entre 0 et 1, delta-temps entre deux calculs dt.

Marche générale

Diffuser la vélocité en fonction de la viscosité du liquide.

En pratique ici, nous n'utilisons pas vraiment cette étape car on estime que l'eau a une viscosité nulle (= moins de temps de simulation pour nous). Mais si on souhaite pousser un peu plus loin cette simulation, il faudra l'utiliser.

L'idée est que si on a un courant continu au milieu de notre environnement, on peut intuitivement comprendre que les molécules d'eau alentour seront influencée, et après un (long) moment, tout l'environnement aura un courant uniforme. C'est la diffusion.

On définit le facteur de diffusion $a = viscosité \times dt$

A chaque instant, 4 parts de la vélocité de la cellule courante se transmet à ses 4 voisins directs et reçoit une part de ces 4 mêmes voisins. (Pour la 3D, remplacer « 4 » par « 6 »)

On obtient alors

$$\vec{v}(x,y) = \overrightarrow{v_{prec}}(x,y) + a \times [\vec{v}(x_{-1},y) + \vec{v}(x_{+1},y) + \vec{v}(x,y_{-1}) + \vec{v}(x,y_{+1}) - 4 \times \vec{v}(x,y)]$$

Mais, bien sûr, il faut connaitre toutes les valeurs de la grille $\vec{v}$ pour calculer $\vec{v}(x,y)$. Pour résoudre ce problème, on peut utiliser un solveur itératif pour déterminer toutes les valeurs de $\vec{v}$ (On parlera des solveurs utilisés/utilisables dans une prochaine partie).

Le grand problème rencontré dans cette équation est qu'elle n'est pas stable pour des valeurs a un peu trop grandes. Ce souci apparait à cause de la partie « $a \times [-4 \times \vec{v}(x,y)]$ », qui peut causer des alternances de valeurs positives-négatives d'une frame à l'autre.

La solution proposée par Jos Stam dans ses papiers *Stable Fluids* (1999) et *Real-time fluid dynamics for games* (2003) est de résoudre l'équation à l'envers. À partir de la même équation retournée :

$$\overrightarrow{v_{prec}}(x,y) = \vec{v}(x,y) - a \times [\vec{v}(x_{-1},y) + \vec{v}(x_{+1},y) + \vec{v}(x,y_{-1}) + \vec{v}(x,y_{+1}) - 4 \times \vec{v}(x,y)]$$

Il propose d'adapter cette équation pour devenir celle-ci (Je n'ai malheureusement pas compris exactement le passage de l'équation précédente à celle-ci) :

$$\vec{v}(x,y) = \frac{\overrightarrow{v_{prec}}(x,y) + a \times [\vec{v}(x_{-1},y) + \vec{v}(x_{+1},y) + \vec{v}(x,y_{-1}) + \vec{v}(x,y_{+1})]}{1 + 4 \times a}$$

(En remplaçant « $1 + 4 \times a$ » en « $1 + 6 \times a$ » pour la 3D)

On remarque ainsi que toutes les composantes de l'équations sont additionnées, sans soustraction. On a alors une consistance entre les frames, peu importe les valeurs de dt et de viscosité.

On ne peut pas trouver les valeurs exactes du champ vectoriel $\vec{v}$, mais on va se rapprocher de la solution en utilisant un solveur itératif afin de réduire l'erreur.

De notre côté, on a considéré la viscosité nulle (donc $a = 0$), donc cette étape peut être ignorée.

Résolution de l'incompressibilité du fluide (« projection »)

On connait le champ vectoriel des vélocités. On peut calculer la divergence des vecteurs à chaque point de la matrice.

Si une cellule voit son vecteur nul et celui de ses voisins se diriger (convergent) vers lui, les fluides viennent vers la cellule et n'en sortent pas, cela se traduit par une forte pression du liquide dans cette cellule. À l'inverse, si les vecteurs sont divergeant, le fluide sort et rien n'en rentre, et alors la pression est faible.

La divergence est définie mathématiquement comme : $div(\vec{A}) = \frac{\partial A_x}{\partial x} + \frac{\partial A_y}{\partial y} + \begin{cases} \frac{\partial A_z}{\partial z} \; en \; 3D \\ 0 \; en \; 2D \end{cases}$

Informatiquement, cela se calcule en chaque point d'une grille de vecteurs (2D) g par

$$div(g(x,y)) = [g(x_{+1},y).x - g(x_{-1},y).x] + [g(x,y_{+1}).y - g(x,y_{-1}).y]$$

On sait que notre solide est censé être incompressible, donc qu'il n'y ait pas de divergence du tout (et donc pas de changement de pression = gradient nul)

Formellement, cela se traduit par l'équation $\vec{\nabla} \cdot \vec{V} = 0$

On va alors calculer nos vélocités $\vec{v}$ puis soustraire une seconde couche de vecteurs $\vec{w}$ tel que $div(\vec{v} - \vec{w}) = 0$. Cette soustraction de vecteurs ($\vec{v} - \vec{w}$) sera notre grille de vélocités « divergence-free » : $\vec{v}_{final}$.

On sait que cette divergence induit un changement de pression (ou gradient de pression ∇p) d'une cellule à l'autre, donc on cherche à résoudre $div(\vec{v}_{final}) = 0$, et par décomposition : $div(\vec{v} - \nabla p) = 0$. Une propriété de la divergence est que $div(a + b) = div(a) + div(b)$.

Notre équation à résoudre ici est alors :

$$div(\vec{v}) = div(\nabla p)$$

(Pour info, la succession d'une divergence de gradient est un operateur Laplacien, noté Δp ou $\vec{\nabla}^2 p$. Mais on va ignorer la notation ici).

On peut facilement calculer le Laplacien en connaissant les valeurs de p :

$$div(\nabla p)[x, y] = p(x_{-1}, y) + p(x_{+1}, y) + p(x, y_{-1}) + p(x, y_{+1}) - 4 \times p(x, y)$$

Malheureusement on ne connait pas les valeurs de p, mais on sait qu'on doit résoudre l'équation :

$$div(\vec{v}) = div(\nabla p)$$

$$div(\vec{v})[x, y] = p(x_{-1}, y) + p(x_{+1}, y) + p(x, y_{-1}) + p(x, y_{+1}) - 4 \times p(x, y)$$

Comme pour la diffusion, on va retrouver les valeurs de pression p à l'aide d'un solveur itératif.

On trouvera enfin notre champ vectoriel de vélocités finales privées de divergences $\vec{v}_{final} = \vec{v} - \nabla p$

Circulation (« advection »)

C'est tout simplement le déplacement de nos particules. Cela se traduit par la vélocité qui se déplace dans la direction dans laquelle elle pointe. Dans mes explications, on va voir chaque vélocité comme si c'était de la matière. Ce n'est bien sûr pas le cas, mais c'est plus simple de le voir comme ça.

La manière naïve de réaliser ce déplacement depuis la cellule (x, y) est de voir sur quelle cellule sa vélocité $\vec{v}(x, y)$ pointe (la cellule $(x + \vec{v}.x, y + \vec{v}.y)$) et de définir la nouvelle vélocité égale à $\vec{v}(x, y)$.

Parce qu'on utilise une grille, quand cette vélocité se déplace, elle se répartie sur les quelques cases autour, pas à une seule case. C'est même une obligation de travailler comme ça parce que si la vélocité est un peu faible, le vecteur ne sortira pas de la cellule, et donc la simulation s'en trouve bloquée. On réalise alors simplement une interpolation linéaire entre le centre des 4 cellules les plus proches (en 2D) ou des 8 cellules en 3D.

Solveur d'équations utilisé dans la diffusion et la projection

Un solveur est utilisé lors des phases de diffusion et de projection dans la simulation de fluide. Dans ces 2 cas, la valeur dans la cellule courante (vélocité ou pression) est dépendante de la valeur des cellules voisines au même instant t.

Lors de la diffusion, quand on cherche à retrouver la vélocité $\vec{v}$ dans chaque cellule, ce qui se traduit par :

$$\vec{v}(x, y) = \frac{\overrightarrow{v_{prec}}(x, y) + a \times [\vec{v}(x_{-1}, y) + \vec{v}(x_{+1}, y) + \vec{v}(x, y_{-1}) + \vec{v}(x, y_{+1})]}{1 + 4 \times a}$$

Idem lors de l'étape de projection, on souhaite retrouver la pression p :

$$p(x_{-1}, y) + p(x_{+1}, y) + p(x, y_{-1}) + p(x, y_{+1}) - 4 \times p(x, y) = div(\vec{v})[x, y]$$

On peut transformer tout cela en un système d'équations linéaires du type

$$\begin{cases} \vec{v}(x,y) = \dfrac{\overrightarrow{v_{prec}}(x,y) + a \times [\vec{v}(x_{-1},y) + \vec{v}(x_{+1},y) + \vec{v}(x,y_{-1}) + \vec{v}(x,y_{+1})]}{1+4\times a} \\ \vec{v}(x_{+1},y) = \dfrac{\overrightarrow{v_{prec}}(x_{+1},y) + a \times [\vec{v}(x,y) + \vec{v}(x_{+2},y) + \vec{v}(x_{+1},y_{-1}) + \vec{v}(x_{+1},y_{+1})]}{1+4\times a} \\ \dots \end{cases}$$

Pour la suite des explications, je prendrai un exemple (aléatoire) plus simple :

$$\begin{cases} x = y + 1 \\ y = \dfrac{x}{5} + \dfrac{2}{5} \end{cases}$$

Chaque équation est dépendante des autres, et il y a trop d'inconnues pour résoudre réellement le système.

Il me semble que l'on est censés pouvoir transformer ce système en matrices sous la forme $\vec{A}\vec{x} = \vec{b}$. Pour être honnête, je ne sais pas comment cela se traduit ici, donc j'ai utilisé un solveur itératif pour résoudre ces systèmes.

Dans le papier de Jos Stam, on utilise la Relaxation de Gauss-Seidel. Concrètement, on initialise toute la matrice à une valeur aléatoire (par convention, l'élément nul [0 pour les nombres, ou $\vec{0}$ pour les vecteurs]). On calcule toutes les équations en utilisant ces valeurs nulles. Par exemple :

$$\begin{cases} x = y + 1 = 0 + 1 = 1 \\ y = \dfrac{x}{5} + \dfrac{2}{5} = 0 + \dfrac{2}{5} = 0.4 \end{cases}$$

Maintenant on a de nouvelles valeurs pour toutes la matrice $(x \quad y \quad z)^T$. On répète l'opération avec ces nouvelles valeurs jusqu'à trouver une convergence, ou un nombre défini de fois.

$$\begin{cases} x = 0.4 + 1 = 1.4 \\ y = \dfrac{1}{5} + \dfrac{2}{5} = 0.6 \end{cases}$$

$$\begin{cases} x = 0.6 + 1 = 1.6 \\ y = \dfrac{1.4}{5} + \dfrac{2}{5} = 0.68 \end{cases}$$

Etc… Après quelques itérations, on converge sur une solution approchée de $x = 1.72$ et $y = 0.736$

D'autres solveurs existent, notamment Jacobi, SOR (Succession OverRelaxations), Multigrid. Je ne connais pas assez ces méthodes pour les décrire ici, mais la différence de l'un à l'autre est généralement le nombre d'itérations avec une convergence, ou la précision de la convergence.

Il est aussi à noter que la relaxation de Gauss-Seidel demande de calculer toutes les équations pour passer d'une itération à l'autre, tandis que la méthode de Jacobi, qui y ressemble beaucoup, peut être divisé en plusieurs threads ou être calculé par la GPU. (Harris, *Fast Fluid Dynamics Simulation* 2004 in GPU gems 3, chapter 38)

Ressources

L'algorithme utilisé est tiré de

Stam, J. (2003). Real-Time Fluid Dynamics for Games. *Proceedings of the Game Developer Conference*, *18*(11), 17. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.12.6736&rep=rep1&type=pdf

Une explications plus poussée et illustrée peut être trouvée sur Medium.com par Shahriar Shahrabi: https://shahriyarshahrabi.medium.com/gentle-introduction-to-fluid-simulation-for-programmers-and-technical-artists-7c0045c40bac

Pour les algorithmes PIC et FLIP avec la méthode Lagrangienne :

Tskhakaya, D., Matyash, K., Schneider, R., & Taccogna, F. (2007). The Particle-In-Cell Method. *Contributions to Plasma Physics*, *47*(8-9), 563-594.

Brackbill, J. U., Kothe, D. B., & Ruppel, H. M. (1988). FLIP: a low-dissipation, particle-in-cell method for fluid flow. *Computer Physics Communications*, *48*(1), 25-38.

Pour l'utilisation du GPU dans cette implémentation de simulation de fluides :

Harris, *Fast Fluid Dynamics Simulation* 2004 in GPU gems 3, chapter 38