Simulation de Fumée

Nicolas Schmitt
nico.schmitt@iquebec.com

Abstract

Dans cet article, nous proposons une méthode pour simuler de la fumée d'une manière réaliste et efficace. Notre méthode se base sur les équations de Navier-Stokes qui régissent les mouvements de la fumée et va donc donner des résultats très proches de la réalité.

De plus, notre système va permettre d'intéragir avec la modélisation en temps réel.

Keywords : *Fluids, Smoke, Navier-Stokes equations, Stable solver, Volume Rendering.*

1 Introduction

La modélisation des phénomènes réels et plus encore la modélisation et le rendu de fluides a toujours été un challenge important en infographie. Ce type de phénomènes a des applications importantes dans l'industrie du jeux vidéo par exemple, mais aussi dans celle des films de synthèse.

Le but de ce projet est le rendu de fumée selon le principe décrit par Stam [1, 2]. Dans son article, il propose une façon relativement simple d'effectuer le rendu de fumée en temps réel. Nous parlerons aussi d'améliorations publiées dans [3].

De plus, la plupart des méthodes que nous allons voir peuvent s'appliquer pour la modélisation d'autre fluides tel que l'eau ou le feu.

Nous allons d'abord voir les lois qui régissent le comportement de la fumée. Ensuite nous allons proposer une méthode simple et robuste pour résoudre ces équations.

Enfin, nous allons voir l'implémentation proprement dite ainsi que le rendu. Notons que le rendu proposé ici est assez simple, une méthode plus complexe intégrée dans un ray-tracer donnerait de meilleur résultats [3]. Néanmoins, le but de ce projet était tout d'abord d'obtenir un système de modélisation complet en 2D et 3D permettant d'effectuer le rendu de fumée en un temps acceptable.

Finalement, nous allons présenter quelques résultats que nous avons obtenus.

2 Previous Works

Jusqu'à présent, la plupart des modèles utilisés pour le rendu de fumée ne recherchaient que la rapidité et un aspect visuel acceptable au dépit du réalisme. La plupart des systèmes utilisés dans les jeux vidéos, par exemple, utilisent les systèmes de particules pour faire le rendu de fumée. Ces techniques, bien que très rapides, ne parviennent pas à modéliser correctement la complexité des fluides.

Foster et Metaxas [4, 5, 6] ont obtenus les premiers résultats intéressant en 1997. Néanmoins, leur méthode de résolution était assez complexe (en $O(n^3)$), et avait l'inconvénient majeur de ne pas être stable. En effet, il fallait que le pas de temps soit petit par rapport aux forces appliquées sur le système. Un autre inconvénient était les dissipations numériques importantes lors de la résolution.

Grâce au système proposé par Stam [1], il est devenu possible d'effectuer le rendu réaliste de fluides en un temps "raisonnable". De plus, son système est stable et robuste, ce qui correspond à une avancé considérable par rapport à Foster.

D'autres travaux, basés sur ceux de Stam, ont depuis amélioré ce système pour tenir compte des spécificités de la fumée [3] ou des liquides [7].

3 Navier Stokes

Afin d'effectuer le rendu de la fumée, il faut résoudre les équations de Navier-Stokes. En notant u la vitesse et ρ la densité, la fumée satisfait les équations :

$$\nabla \cdot u = 0 \tag{1}$$

$$\frac{\partial u}{\partial t} = -(u \cdot \Delta)u + \nu\Delta^2 u + f \tag{2}$$

$$\frac{\partial \rho}{\partial t} = -(u \cdot \Delta)\rho + \kappa\Delta^2 \rho + S \tag{3}$$

où f et S sont respectivement les forces appliquées au modèle et les sources de matière.

Dans notre modèle, nous avons inclus la température. Cette dernière satisfait l'équation :

$$\frac{\partial T}{\partial t} = -(u \cdot \Delta)T + S_c \tag{4}$$

avec S_c les sources de chaleur (ou de froid).

La température et la densité vont également changer le champ des vitesses. Ces actions seront modélisées par une force :

$$\vec{f} = -\alpha\rho\vec{z} + \beta(T - T_{amb})\vec{z} \tag{5}$$

où α et β sont deux constantes positives rendant cette équation physiquement acceptable.

Il va donc falloir résoudre ces équations pour un pas de temps Δt donné. Notons qu'en pratique, l'équation (3) se réduit à :

$$\frac{\partial \rho}{\partial t} = -(u \cdot \Delta)\rho + S \tag{6}$$

Notons également que cette équation va s'appliquer à toute substance qui se déplace dans le champ de vecteur. En pratique, on va appliquer cette équation à la température.

Nous avons également appliqué cette équation à des coordonnées de textures, les résultats sont assez intéressants.

4 Méthode de Résolution

Notons tout d'abord que la modélisation que nous présentons ici est valable dans le cas de la 2D comme dans le cas de la 3D.

La première étape consiste à discrétiser l'espace et le temps. Le principe proposé par Stam étant stable, la valeur du pas de temps ne devrait pas avoir d'importance sur l'obtention d'une solution acceptable.

Pour la discrétisation de l'espace, nous l'avons divisé en une grille de voxels. Les températures et les densités sont définies au centre de chaque cellule.

Nous avons également choisi de définir le champ de vitesse au centre des voxels comme dans [1], mais à la différence de [3] et [4, 5, 6].

On voit alors que l'équation (2) se décompose en trois parties qui seront résolus par les trois partis majeures de notre solveur. Notons également qu'une fois la première équation résolue, les deux autres équations peuvent se résoudre avec les mêmes méthodes.

Notons également que les solutions obtenues vont dépendre grandement des conditions limites que nous avons choisis, nous allons présenter la résolution dans le cas de conditions limites périodiques, c'est à dire que l'espace se referme sur lui-même. Nous présenterons plus loin dans cet article les modifications à apporter pour considérer des conditions aux bornes plus intéressantes.

La première partie dont nous allons nous occuper est le terme de self-advection : $-(u \cdot \Delta)u$.

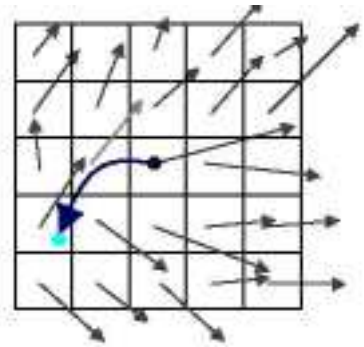
Ce terme est celui qui pose le plus de problème car il est non-linéaire. Pour le résoudre, non allons utiliser une méthode semi-lagrangienne, qui à l'inverse de la méthode de Foster et Metaxas, va rester stable quelque soit le pas de temps utilisé.

Le principe de cette méthode est de parcourir la grille de densité, et pour chaque voxel, on va chercher les particules qui vont arriver au centre de cette cellule après un temps Δt. Pour cela, on va partir de l'arrivé et appliquer les forces courantes pendant un temps $-\Delta$t. On va donc arriver à l'endroit qui après Δt sera au centre du voxel considéré. Malheureusement, ce point est un endroit quelconque de la

grille (i.e. pas le centre d'un voxel), on va alors faire une interpolation basée sur les quatre cellules les plus proches. La valeur que l'on va trouver va alors être placée dans le voxel de départ.

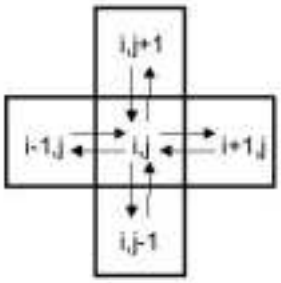
Cette méthode ne va donc pas se limiter au déplacement entre cellule voisine comme ce serait le cas avec une approche classique en différences finies.

Le schéma suivant montre cette étape :



Nous allons maintenant montrer comment résoudre le deuxième terme de l'équation (2) : $\Delta^2 u$.

Ce terme rend compte de la diffusion, c'est à dire les échanges de matières entre cellules de la grille. L'approche classique consiste à ne considérer que les échanges entre voxel voisins.



Encore une fois, l'approche inverse consistant à remonter le temps va être plus stable et va permettre de considérer des échanges supérieurs à ceux du premier ordre.

En pratique, cette méthode va consister à résoudre un système linéaire, ce qui va coûter très cher (voir le chapitre sur les conditions limites non périodiques).

Dans le cas de conditions limites périodiques, ce terme sera résolu dans l'espace de Fourier. En effet, dans l'espace de Fourier, les fluides ont des particularités très intéressantes.

En effet, dans cet espace magique, le terme de diffusion n'est en fait qu'un filtre passe bas. Afin de tenir compte de la constante de diffusion κ, nous avons choisi comme filtre :

$$\exp(-k^2 \kappa dt)$$

Où k est le nombre d'onde, c'est à dire la distance à l'origine dans l'espace de Fourier.

Une autre propriété des fluides est la conservation de la masse, donné par (1). En pratique, cela veut dire que les flux entrant et sortant sont égaux. Une fois encore, cela va aboutir à un système linéaire (i.e. coûteux à résoudre...).

Dans le cas où l'espace est périodique, nous allons à nouveau utiliser les propriétés des fluides dans l'espace de Fourier. Nous allons nous servir de la décomposition de Helmholtz qui dit que tout champ de vecteur est la somme d'un champ respectant la conservation de la masse et du champ gradient. On a en plus la propriété que le champ gradient à tous ses vecteurs parallèles au nombre d'onde, l'autre champ ayant ses vecteurs perpendiculaires.

Pour obtenir un champ de vecteur respectant la conservation de la masse, il suffit donc de projeter les vecteurs de l'espace de Fourier sur les cercles centrés à l'origine.

Nous avons donc montré comment résoudre deux des termes de l'équation (2). Le dernier terme est simplement l'application des forces, c'est à dire que l'on ajoute à la vitesse la force multipliée par Δt.

5 Forces de Confinements

La discrétisation de l'espace-temps et les méthodes numériques pour résoudre les équations vont ajouter des dissipations dans nos calculs. En fait, le comportement de notre fluide va tendre vers le comportement réel quand la taille d'un voxel et que le pas de temps tendent vers 0.

Néanmoins, afin d'améliorer l'aspect visuel, une technique consiste à réinjecter des forces dans le modèle. La méthode que nous allons utiliser est présentée dans [3] et donne des résultats intéressants.

Visuellement, les forces de confinement vont faire tourner localement le fluide.

Les forces de confinements sont données par :

$$\omega = \nabla \times u \tag{7}$$

$$\eta = \nabla |\omega| \tag{8}$$

$$N = \frac{\eta}{|\eta|} \tag{9}$$

$$f_{conf} = \epsilon h (N \times \omega) \tag{10}$$

où h est la taille d'un voxel et ϵ est une constante positive contrôlant la quantité de détails que nous voulons réinjecter dans la simulation.

6 Implémentation

L'implémentation à été réalisé en `C++` sous Windows, et a été testé avec succès sous Linux avec Wine [8]. Pour le chargement et la sauvegarde des images, nous avons utilisé DevIL [9].

Comme nous l'avons vu précédemment, certain calculs sont effectués dans le domaine fréquentiel, d'autre dans le domaine spatial, il nous faut donc un moyen efficace de passer de l'un à l'autre. Nous avons choisis d'utiliser la librairie FFTW [10] qui offre une vitesse intéressante (au détriment de la simplicité des structures de données). Le schéma (1) nous donne une vue globale, le rectangle grisé nous montre les opérations réalisées dans le domaine fréquentiel.

Afin de pouvoir appliquer la méthode semi-lagrangienne, il va nous falloir deux tableaux par champs scalaires, soit deux pour les densités, deux pour les températures, et $2N$, avec N le nombre de dimensions, pour les forces. De plus, si on utilise le vorticity confinment, il va nous falloir $N+1$ tableaux supplémentaire. Dans le cas d'une simulation 3D basée sur une grille de $80 \times 80 \times 80$, ces structures occupent environ 85Mo.

La structure globale du programme est :

```
void calcul ( void )
{
  SwapGrids();
  // Les forces
  for ( i = 0; i < NDIM; i++ ) {
    AddForce ( u0[i], f[i] );
  }
  for ( i = 0; i < NDIM; i++ ) {
    Transport ( u1[i], u0[i], u0 );
  }
  GoIntoFourier();
  for ( i = 0; i < NDIM; i++ ) {
    DiffuseFFT ( u0[i], visc );
  }
  ProjectFFT ( calculs.u0 );
  BackFromFourier();
  // Densité
  Transport ( s1, s0, u1 );
  Dissipate ( s1, aS );
  // Température
  AddForce ( t0, chaleur );
  Transport ( t1, t0, u1 );
  Dissipate ( t1, 0.8 );
}
```

Comme nous l'avons vu précédemment, la fonction `DiffuseFFT` est simplement un filtre passe-bas. La fonction `ProjectFFT` va projeter les vitesses sur les cercles centrés à l'origine.

La fonction `Transport` a la structure suivante :

```
Transport ( s1, s0, U ) {
  pour tous les voxels {
   TraceParticle ( U, -dt );
   s1 = LinInterp ( s0 );
  }
}
```

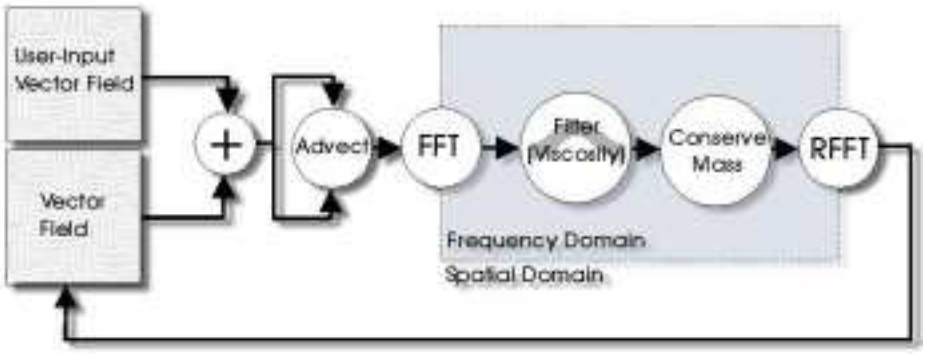


FIG. 1 – Structure du Programme

7 Le Rendu

Le rendu a été fait sous OpenGL en utilisant la primitive TRIANGLE_STRIP. Chaque voxel est affiché grâce à deux triangles. Pour l'affichage du champ de vitesse, on affiche pour chaque voxel, le vecteur de vitesse. Notons que cette technique pour afficher les vitesses donne de bons résultats en 2D mais des résultats moins utilisables en 3D.

En 2D, l'affichage est fait tout simplement en mode orthographique. Dans le cas de la 3D, on effectue un "splatting maison". En fait, on va afficher les grilles 2D pour chaque profondeur en utilisant le blending. Notons qu'il est alors important d'afficher les voxels d'avant en arrière.

La couleur de chaque sommet des triangles sera la densité au point considéré.

Le rendu que nous avons utilisé ici permet d'obtenir des vitesses de plusieurs dizaines d'images par secondes en 2D. En 3D, la vitesse va tomber à environ 10 secondes par images.

Afin d'ajouter du réalisme à la scène, nous avons pensé à ajouter des ombres au rendu. Pour cela, les shadow maps ont été choisis, et bien que la partie documentation est été finie, le temps a manqué pour l'implémentation.

8 Résultats

Nous allons présenter ici les résultats que nous avons obtenues avec notre implémentation. Les temps ont été mesurés sur un Pentium4 1Ghz sous Windows.

En 2D, pour un quadrillage de 80×80 voxels, la simulation se fait en temps réel (40 images par secondes). En 3D, pour un quadrillage de $80 \times 80 \times 80$ voxels, il faut environ 10 secondes par images.

Pour les constantes, nous avons choisis :

$$\Delta t = 0.4$$

$$h = 1$$

$$visc = 0.001$$

$$aS = 0.90$$

La figure (2) nous montre certain de nos résultats. Dans la dernière image, nous avons placé une boule qui projette de la fumée horizontalement. Pour rendre la simulation plus réaliste, nous injectons continuellement des forces aléatoires dans le modèle.

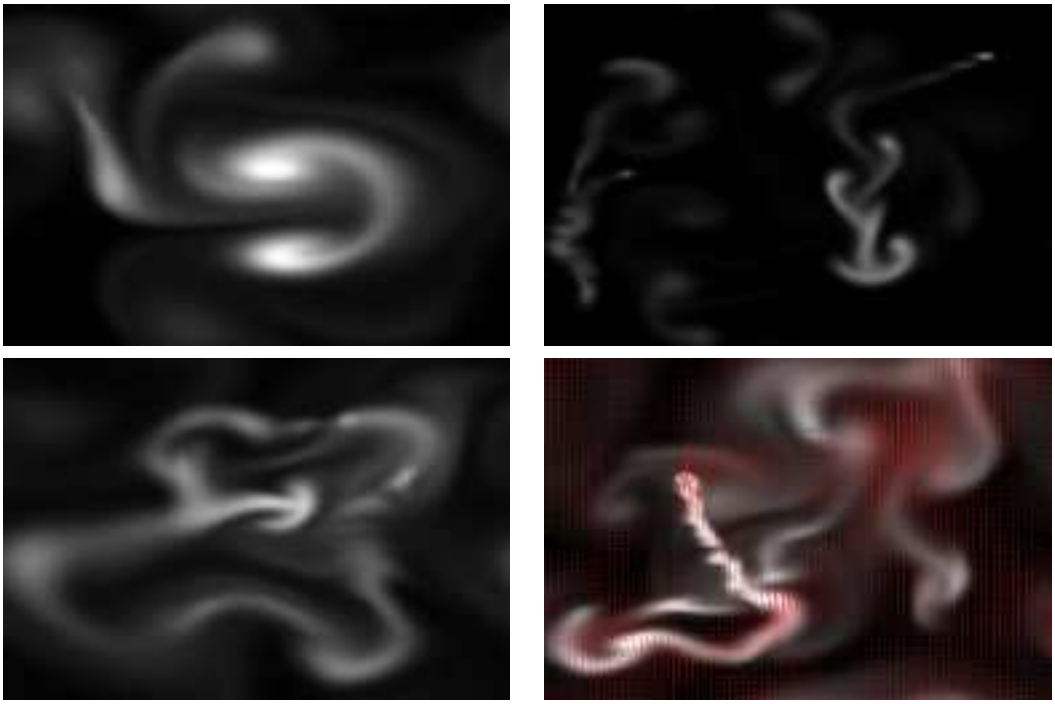


FIG. 2 – Quelques exemples en 2D

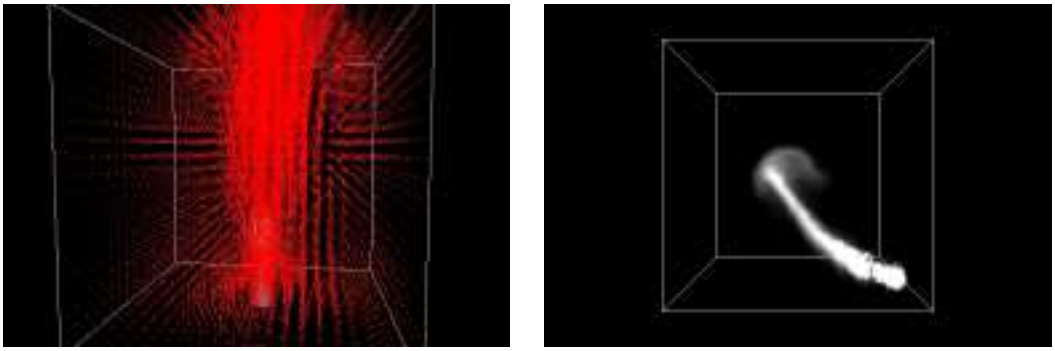


FIG. 3 – A droite, un canon de fumée. A gauche on à affiché les vitesses

9 Cas des conditions limites non périodique

Dans le cas on l'on ne considère plus l'espace comme étant périodique, l'approche utilisant la FFT ne peux plus être appliquée. Dans ce cas, les fonctions `Diffuse` et `Project` sont beaucoup plus compliquées. Comme nous l'avons dit plus tôt, il est nécessaire de résoudre un système linéaire assez important à chaque fois.

Le tableau suivant présente les différent systèmes que l'on peut utiliser :

Nom	Coût
Gaussian Elimination	N^3
Jacobi / SOR	N^2
FFT	$N \log N$
Gradient Conjugué	$N^{1.5}$
Multigrid	N

TAB. 1 – Méthodes de résolution

Théoriquement, la méthode multigrid est la plus rapide, mais en pratique, elle s'avère très (très) dure à coder. Nous avons choisis d'utiliser une résolution basée sur le gradient conjugué. Le code que nous avons écrit se base sur [11]. Malheureusement, cette méthode n'a pas convergé dans notre implémentation. Une raison pourrait être l'absence de pré-conditionnement (qui est plus dure à implémenter).

Néanmoins, cette méthode pourrait être intéressante à utiliser car elle autorise des conditions limites plus réelles (comme des murs par exemple), et la présence d'objets dans la scène (voir [3] pour des exemples).

10 Améliorations

Avec plus de temps, plusieurs améliorations auraient été envisageables.

Tout d'abord, au niveau de la modélisation, il a été montré que le vorticity confinement améliorait l'aspect visuel de la fumée. D'autre type de force pourrait être envisagées afin de corriger les dissipations dues au calculs numériques et à la discrétisation.

De plus, il pourrait être intéressant de ne plus se limiter aux conditions limites périodiques. Avec cette méthode il serait théoriquement possible d'insérer des objets dans la scène, ce qui pourrait donner des résultats très convaincants (voir [3]).

Au niveau du rendu, seul une méthode a été abordée. De plus, cette technique bien que rapide ne donne pas des résultats aussi bon qu'en utilisant le ray-tracing. Mais cette technique nécessite tout d'abord de programmer un ray-tracer, et deuxièmement de le modifier pour intégrer le rendu volumique, ce qui aurait demandé un travail trop important.

Un autre moyen d'améliorer le rendu est l'intégration d'ombre dans le rendu. Une technique utilisant les shadow maps est en cours de codage, mais le temps nous a manqué pour la finir.

11 Conclusion

Nous avons donc présenté une méthode pour simuler de la fumée en un temps raisonnable. Cette approche se base sur les équations réelles de la mécanique des fluides, notre implémentation donne donc des résultats physiquement exacts.

De plus, grâce à l'utilisation des forces de confinement, notre simulation ne souffre pas de dissipations dues au calculs numériques.

Afin d'améliorer encore la rapidité, il pourrait être envisageable de supprimer le passage du domaine de Fourier au domaine spatial, pour cela, il faudrait trouver un moyen de réaliser la méthode semi-lagrangienne dans le domaine de Fourier, mais cette idée ne me parait pas avoir de solution simple et est donc encore du domaine de la recherche...

Références

[1] Stam J., Stable Fluids, In *Computer Graphics Proceedings, Annual Conference Series, 1999.*

[2] Stam J., A Simple Fluid Solver based on the FFT, *SIGGRAPH, 2000.*

[3] Fedkiv, Stam et Jensen, Visual Simulation of Smoke, *SIGGRAPH 2001.*

[4] N. Foster and D. Metaxas. Realistic Animation of Liquids. *Graphical Models and Image Processing, 1996.*

[5] N. Foster and D. Metaxas. Modeling the Motion of a Hot Turbulent Gas. *Computer Graphics Proceedings, Annual Conference Series, 1997.*

[6] N. Foster and D. Metaxas. Controlling Fluid Animation. *Proceeding of the Computer Graphics International, 1997.*

[7] N. Foster, R. Fedkiw. Practical Animation of Liquids, *SIGGRAPH 2001.*

[8] Wine : http ://www.winehq.com

[9] The Developer's Image Library (DevIL) at http ://www.imagelib.org

[10] Fastest fourier transform in the west, http ://www.fftw.org

[11] J.R. Shewchuk, An introduction to the Conjugate Gradient Method Without the Agonizing Pain, 1994