Projet de spécialité 2010

Conception d'un modèle de feu 3D temps réel

Etudiants impliqués :
Benjamin Aupetit - IRVM - benjamin.aupetit@ensimag.imag.fr
Julien Champeau - IRVM - julien.champeau@ensimag.imag.fr
Arnaud Emilien - IRVM - arnaud.emilien@ensimag.imag.fr

Encadrants :
Marie-Paule Cani - Marie-Paule.Cani@inrialpes.fr
Aurélie Catel - aurelie.catel@grenoble-inp.fr

Ensimag 2010

Table des matières

1 Présentation du projet

1.1 Introduction

1.2 Objectifs initiaux

Le but principal de notre projet est de modéliser du feu en temps réel. Cette modélisation devra être la plus réaliste possible, en effet nous voulons permettre d'integrer notre solution dans un environnement temps réel qui requiert un comportement proche de la réalité en restant intéractif. Dans ce sens notre solution ne sera pas adaptée à tous les jeux videos, car ceux ci ne necessitent pas une approche réaliste mais juste des effets visuels impressionants.

Notre solution devra ainsi répondre aux critères suivants :
- Modèle de flamme convainquant
- Génération de fumée
- Propagation réaliste sur l'objet
- Prise en compte des flux d'air (vent, advection du feu...)
- Destruction procedurale progressive d'un objet
- Propagation à l'environnement

1.3 Aperçu

2 Aperçu de notre modèle

Ici nous présentons un résumé de notre modèle global. Ce modèle a été établi pour avoir un idée générale et ainsi choisir les modèles adaptés pour chaque parties. En effet nous devons nous assurer de la compatibilité de ceux ci, pour eviter les incohérences entre les différentes étapes, ou des calculs de syncrhonisation entre les phases. Par exemple la représentation des objets doit être définié dès le début mais aussi permettre toutes les opérations que nous pourrions être amenné à y apporter.

2.1 Les flammes

Pour modéliser les flammes il faut différencier deux aspects. Le rendu des flammes d'une part, et la modélisation du phénomène associé d'autre part.

2.1.1 Un modèle de fluide pour le feu

Basé sur les travaux de **Jos Stam** (stable fluids). Ce modèle inclu la diffusion de la densité, la chaleur, la vitesse des fluides utilisés ("combustible", fumée, air). Il permettra aussi de gerer plus simplement la diffusion du feu dans l'environnement.

2.1.2 Le rendu des flammes et de la fumée

Pour rendre les flammes nous avons besoin d'une méthode rapide mais donnant une impression de réalisme. Pour faire ceci nous utilisons une méthode expliquée **Zeki Melek**. Cette méthode consiste à découper le fluide en "tranches" othogonale à l'orientation de la camera. Dans chaque tranche on calcule

une texture d'après la densité de fluide représentant la flamme. Ensuite avant le rendu on applique un bruit de Perlin (en 4D) sur chaque tranche, et enfin on applique un effet de "bloom" sur chaque texture pour modéliser la luminosité du feu. De plus nous appliquons une couleur à la flamme qui est fonction de sa chaleur (principe du corps noir).
La fumée sera rendue de la même manière que la flamme.

2.2 Les objets

Dans un premier temps nous nous concentrerons sur des objets qui se décomposent lors de la combustion. Pour cela nous utilisons une décomposition tétrahédrale des maillages des objets pour les décomposer. Cette méthode permet en effet de détruire localement les maillages en conservant la géometrie locale des objet (si elle ne brule pas).

3 Analyses Bibliographiques

Cette section regroupe les différents articles lus, concernant les travaux déjà effectués dans ce domaine. Nous allons expliquer brièvement de quoi ils parlent, ce que nous en avons retenu de bien ou de mal, ce que nous allons utiliser...

3.0.1 Real-Time Fluid Dynamics for Games

Auteur(s) : Jos Stam.
Publication : Proceedings of the Game Developer Conference, March 2003
Sujet(s) abordé(s) :
Modèle de fluide temps réel, pouvant être utilisé pour le feu, la fumée
Le modèle gère : le deplacement du fluide, la gestion de combustible,les forces appliquées sur le fluide.
Principe :
Le modèle a été décomposé sur plusieurs étapes : génération du fluide par des sources, ajout des forces, diffusion du fluide, puis résolution de l'équation de la densité et de l'équation de la vitesse (équations de Navier-Stockes). Pour résoudre les deux équations, il utilise une astuce permettant de résoudre le système très rapidement. Enfin, il utilise le principe de conservation de la masse, qui rajoute des effets réalistes de vortex, avec la décomposition de Hodge.
Point(s) positif(s) :
Temps réel.
C'est facile à implémenter (moins de 100 lignes pour la version 2D)
Peut être adapté pour la propagation du feu.
Peut être adapté pour qu'un objet influe sur le modèle.
Résultat réellement joli et réaliste.
Très bien expliqué.
Il a fournit un exemple 2D pour de la fumée, plutôt impréssionant.
Le modèle peut s'adapter au feu, à la fumée, et à l'eau.
Point(s) négatif(s) :
La zone d'influence est assez petite, il faut voir si c'est possible de l'étendre sans trop allourdir les calculs. (Octree ?)
Pas trop de détails sur la représentation graphique du feu.

La dissipation numérique implique que le résultat n'est pas exact.
Conclusion :
C'est sans doute ce que nous allons adapter, pour qu'il serve à la fois pour le feu, la fumée, et la propagation.

3.0.2 Stable Fluids

Auteur(s) : Jos Stam.
Publication : SIGGRAPH 99 Conference Proceedings
Sujet(s) abordé(s) :
Résolution de l'équation des fluides (Navier-Stockes) avec, pour la première fois, un algorithme inconditionnellement stable.
Principe :
C'est une résolution de Navier-Stockes orientée "Computer Graphic", dans le sens o'u la résolution n'est pas exacte, et ne serait pas utilisable pour des vraies simulations de fluides, mais est "graphiquement adaptée" au problème de fluides.
Résolution en quatre étapes : ajout des forces, advection, diffusion, conservation de la masse.
Point(s) positif(s) :
2D et 3D.
Facile à implémenter.
Résolution stable et temps réel.
Résultat réaliste.
Point(s) négatif(s) :
La dissipation numérique implique que le résultat n'est pas exact.
Conclusion :
Ce modèle de fluide semble le plus adapté si nous choisissons d'utiliser un modèle de fluide pour le feu, la fumée et la propagation.

3.0.3 An Interactive Simulation Framework for Burning Objets

Auteur(s) : Zeki Melek, John Keyser.
Publication : Technical Report 2005 3 1, Texas AM Department of Computer Science, 2005.
Sujet(s) abordé(s) :
Premier modèle essayant de réaliser en même temps une propagation de feu réaliste avec un modèle de fluide, de la propagation sur un objet et entre les objets, et de la destruction d'objets.
Principe :
Le modèle de fluide utilise Stable Fluid de **Stam**.
Point(s) positif(s) :
Temps réel
Modèle de fluide, propagation, gestion des objets, destruction des objets.
Point(s) négatif(s) :
L'implémentation CPU est lente (4fps pour une grille 20*20*20)
Conclusion :

3.0.4 Visual Simulation of Smoke

Auteur(s) : Ronald Fedkiw, Jos Stam, Henrik Wann Jensen.
Publication : SIGGRAPH 2001 Conference Proceedings
Sujet(s) abordé(s) :
Création d'un modèle de fumée, basé sur le travail de Stam (Stable Fluids).
Rendu réaliste de la fumée.
Principe :
L'équation du fluide est résolue comme dans "Stable Fluids". Cette fois la chaleur est prise en compte, et traitée de la même manière que la densité. Il en résulte une force de préssion qui s'ajoute simplement au modèle. Le modèle prend en compte les petits phénomènes de vortex qui se créent dans le fluide, basée sur la méthode de Steinhoff ("Vorticity confinement"). Ainsi ils rajoutent une force de rotation pour créer les minis vortex.
Pour le rendu : ils découpent la grille de voxels en plusieurs plans, rendus comme une superposition de plans transparents. Une autre méthode de rendu, plus réaliste, utilise une méthode de lancé de rayons.
Point(s) positif(s) :
Gestion de la chaleur.
Rendu très réaliste de la fumée.
Point(s) négatif(s) :
Pas en temps réel.
Conclusion :
La méthode de rendu semble interessante. De plus la création de minis vortex est un atout pour le côté réaliste du modèle.

3.0.5 Simulating Water and Smoke with an Octree Data Structure

Auteur(s) : Frank Losasso, Frédéric Gibou, Ron Fedkiw.
Publication : SIGGRAPH 2004
Sujet(s) abordé(s) :
Simulation d'eau et de fumée. L'équation de Navier-Stockes est résolue sur une grille Octree.
La grille du modèle de fluide s'adapte selon le niveau de détail du phénomène (par exemple plus de détails là o'u il y a des minis vortex)
Principe :
Le calcul de l'équation des fluides est effectué sur un Octree.
Point(s) positif(s) :
L'octree n'est pas restreint (ce qui était le cas des travaux précédents).
Réduction des calculs de 75% pour un résultat équivalent avec une grille à pas constant.
Point(s) négatif(s) :
Pas temps réel.
Compliqué à mettre en place.
Conclusion :
La complexité de la programmation est assez importante. L'avantage de cette méthode est d'avoir un très haut niveau de détail en effectuant plus de calculs là o'u cela est nécessaire. La précision que nous désiront n'est peut être pas suffisante pour nécessiter un tel modèle.

3.0.6 Real-Time Simulation of Deformation and Fracture of Stiff Materials

Auteur(s) : Matthias Müller, Leonard McMillan, Julie Dorsey, Robert Jagnow.
Publication : EUROGRAPHICS 2001 Computer Animation and Simulation Workshop
Sujet(s) abordé(s) :
Destruction réaliste et temps réel d'un objet.
Principe :
C'est la simplification d'un problème de propagation en négligeant les effets microscopiques, qui ne sont pas vraiment visibles en temps réel, mais coûtent énormement en calcul.
Les meshs sont représentés par des "tetrahedral meshes". Le modèle de propagation continu est transformé en modèle discret. L'élasticité des objets est prise en compte.
Point(s) positif(s) :
Temps réel.
Le système est stable est rapide.
La méthode de destruction de l'objet basée sur des tétrahèdres est très intéressante.
Propagation des effets de la destruction à l'intérieur de l'objet.
Point(s) négatif(s) :
Peut être lourd à utiliser si il y a trop d'objets à l'écran.
Conclusion :
Le principe de déformation sur un mesh tétrahédral est très intéressant, et peut facilement s'adapter à un modèle de fluide.

3.0.7 Voxels On Fire

Auteur(s) : Ye Zhao, Xiaoming Wei, Zhe Fan, Arie Kaufman, Hong Qin.
Sujet(s) abordé(s) :
Animation et propagation de flammes, avec brulure de l'objet.
Principe :
L'animation et la propagation sont gérés par un modèle de fluide (Lattice Boltzmann Model), avec une grille régulière. L'objet est représenté sous forme de voxels, sur lequel est calculé un champs de distance. Le voxel de l'objet est considéré comme un combustible. Pour le rendu ils utilisent des particules et le splatting (**Westover**).
Point(s) positif(s) :
- temps réel (24fps pour une grille 64*64*64)
- propagation réaliste
- l'objet est brulé progressivement
Point(s) négatif(s) :
- le rendu utilise un grandn ombre de particules pour être réaliste
- l'objet n'est pas détruit seule la texture est altérée
Conclusion :

3.0.8 Meshes On Fire

Auteur(s) : Haeyoung Lee, Laehyun, Mark Meyer, Mathieu Desbrun.
Publication : Eurographics Workshop on Computer Animation and Simulation '2001
Sujet(s) abordé(s) :
Propagation des flammes à la surface d'un objet.
Principe :
La propagation est un parcours géodésique de la surface, et subit le vent environnant.
Les flammes sont rendues sont formes de particules avec des blobs.
Les flammes générées subissent le champ de vent ce qui les rend plus réaliste.
Point(s) positif(s) :
Temps réel.
Prise en compte de plusieurs feu.
Rapide à calculer.
Très efficace pour changer la texture d'un objet brulé.
Propagation fonction d'un champ de vent.
Les particules de feu générées peuvent être utilisées pour génerer un feu sur un autre objet ou sur une autre partie de l'objet.
Point(s) négatif(s) :
Ne peut pas s'adapter à la destruction d'un objet.
Peut être une perte de performance si on utilise les particules de feu pour allumer des foyer aux autres endroits du mesh.
La propagation surfacique n'est pas toujours adaptés.
Conclusion :
Ce modèle est interessant, mais nécessite d'être beaucoup modifié pour correspondre à notre but. Il n'est pas très adapté à la destruction des objets. (Sauf si nous voulons uniquement dégrader la texture de l'objet)

3.0.9 Real-time Procedural Volumetric Fire

Auteur(s) : Alfred R. Fuller, Hari Krishnan, Karim Mahrous, Bernd Hamann, Kenneth I. Joy.
Publication : Proceedings of the 2007 symposium on Interactive 3D graphics and games
Sujet(s) abordé(s) :
Rendu de feu réaliste en temps réel, utilisant le bruit de perlin. **Principe** :
La flamme est calculée en 3 dimensions, et le rendu est effectué par le GPU.
L'animation de la texture est procédurale.
Point(s) positif(s) :
Temps réel.
Réaliste.
Point(s) négatif(s) :
Représentation du feu uniquement, pas de propagation et de fumée.
Pas d'utilisation d'équation de dynamique des fluides, le feu est manuellement confiné dans un volume controlable grace à des points de controle. **Conclusion** :
La qualité du feu est importante, mais ce modèle de flamme ne concerne que la partie "affichage". Il faudra voir si le modèle de propagation choisi peut utiliser

un tel modèle de flammes.

3.0.10 Melting and Burning Solids into Liquids and Gases

Auteur(s) : Frank Losasso, Geoffrey Irving, Eran Guendelman, Ron Fedkiw.
Publication : IEEE TVCG 12, 343-352 (2006).
Sujet(s) abordé(s) :
Modification des propriétés physiques d'un matériau tel que l'eau sous différentes phases, et intéraction entre les solides et les liquides et gas. **Point(s) négatif(s)** :
Peu de détails mathématiques, juste des explications.
Pas assez de lien avec le feu et la fumée. **Conclusion** :
Peu utilisable

3.0.11 FireStarter – A Real-Time Fire Simulator

Auteur(s) : Marc de Kruijf.
Sujet(s) abordé(s) :
Description du feu réel, des différents modèles utilisés, et réalisation d'un modèle avec des particules.
Principe :
Le feu est un système de particules, les flammes sont générées avec une vie aléatoire, une vitesse verticale dépendant de la chaleur, et une vitesse horizontale aléatoire. De plus, un test aléatoire fonction de la durée de vie de la particule est effectué pour qu'elle se transforme en fumée. Le tout est effectué dans une grille cylindrique 3D. Au niveau de l'affichage, les particules sont reliées pour former des faces qui seront ensuite texturées et colorées en fonction de la chaleur.
Point(s) positif(s) :
Temps réel.
Calcul très rapide et très simple.
Point(s) négatif(s) :
Pas très réaliste actuellement. (pourrait peut-être être amélioré par des experts)
De gros défauts sont visibles au niveau du rendu.
Les particules suivent une pseudo mécanique des fluides.
Le feu n'est pas adapté à plusieurs cas de figures, il faudrait un système de génération des générateurs de particules, ce qui n'est pas forcement permis par le système.
Pas de prise en compte des objets (à moins d'ajouter des détections de collisions)
Conclusion :
La bonne idée de transformer une particule de feu en fumée à partir d'un tirage aléatoire est une bonne idée. Ce modèle permet de représenter du feu d'une manière qui pourrait être suffisamment réaliste pour ne pas se lancer dans une résolution de mécanique du fluide.

3.0.12 Simulating Fire With Texture Splats

Auteur(s) : Xiaoming Wei, Wei Li, Klaus Mueller1 and Arie Kaufman.
Publication : IEEE Visualization 2002.

Sujet(s) abordé(s) : Méthode d'affichage de feu avec des particules et une texture "Splat"
Principe :
Point(s) positif(s) :
Conclusion :

3.0.13 Physically Based Modeling and Animation of Fire

Auteur(s) : Duc Quang Nguyen, Ronald Fedkiw, Henrik Wann Jensen.
Publication :
Sujet(s) abordé(s) :
Modélisation du feu comme le comportement d'un fluide suivant les équation de Navier-Stokes de la dynamique des fluide.
Principe :
De même que les articles de Stam, Le principe est de considérer le feu comme un fluide auquel on applique les équations de Navier Stokes pour les fluides incompressibles. On prend aussi en compte la densité du fluide mais aussi l'impact de la température sur celui ci. Cela permet d'avoir un champ de mouvement pour le fluide qui réagit avec la température. De plus l'utilisation de la température pour ce modéle permet d'être utilisé pour le rendu en considérant le feu comme un "corps noir" et de déterminer la coleur de la flamme grace au modéle du corps noir de planck en utilisant un spectre de couleur adapté.
Point(s) positif(s) :
Plus de réalisme dans le modéle grace à l'impact de la température pour l'advection et la gestion du spectre lumineux du feu.
Point(s) negatif(s) :
rajoute du temps de calcul pour la prise en compte de la température.
Conclusion :
Certains points comme la température peuvent être à rajouter pour un effet de réalisme intéressant.

3.0.14 Real-Time Simulation and Rendering of 3D Fluids

Auteur(s) : Keenan Crane, Ignacio Llamas, Sarah Tariq.
Publication : GPU GEMS
Sujet(s) abordé(s) :
Résolution des équations de Navier-Stokes grace a la programmation GPU pour atteindre le temps réel. Les travaux étant réalisé par NVIDIA sur des cartes graphiques performantes et utilisant le shader langage GS ou CUDA.
La Technique de collision entre gas et objets est aussi détaillée ainsi que la voxelisation de certains objets.
Principe :
Utilisation des langages de programmation utilisant les shaders GPU pour permettre un calcul bien plus rapide que par CPU et donc atteindre le temps réel bien plus facilement.
Point(s) positif(s) :
Les temps de calcul atteint par les GPU sont néttement plus faibles que ceux du CPU. On gagne donc a traiter les equations de dynamique des fluides par

la carte graphique et non par le processeur. Les equations de la dynamique des fluides sont traitées de la meme maniere que dans les différents documents de Stam vus précédement.
Point(s) negatif(s) :
Nous n'allons pas utiliser le shading langage du document mais plutot le GLSL (openGL Shading Langage) qui est légérement différent dans la facon de coder. Il nous faut donc chercher un peu plus de documentation sur le codage GLSL.
Conclusion :
L'utilisation des shaders GPU semble être la clé pour nous permettre de réaliser quelquechose de réaliste en temps réel.

3.0.15 Fast Fluid Dynamics Simulation on the GPU

Auteur(s) : Mark J. Harris.
Publication : GPU GEMS
Sujet(s) abordé(s) :
Chapitre du livre GPU Gems de NVIDIA traitant d'une méthode de resolution des équations de "stable fluids" entierement réalisées sur GPU.
Principe :
Tous les calculs pour les champs de vitesses et de densités pour les fluides (diffusion, advection, forces, conservation de la masse) sont réalisé sur GPU pour augmenter la vitesse et la puissance de calcul.
Point(s) positif(s) :
Les termes de l'équation de Navier stokes sont tous décortiqués et trés bien expliqués.
Facilement Compréhensible et tous les calculs mathématiques sont bien détaillés.
Modéle 2D pour que la compréhension soit facile.
Prise en compte d'un terme de pression.
Utilisation de la décomposition de Helmholtz-Hodge
Exemple de programmation GPU (type CUDA ou GS)
Point(s) negatif(s) :
Toujours pas de "cours" GLSL.
Conclusion :
Document qui peut servir de référence pour analyser et comprendre les equations de la dynamique des fluides et pour initier les bases de la programmation GPU.

3.0.16 Reliable Isotropic Tetrahedral Mesh Generation Based On An Advancing Front Method

Auteur(s) : Yasushi Ito, Alan M. Shih and Bharat K. Soni.
Publication :
Sujet(s) abordé(s) :
Principe :
Point(s) positif(s) :
Point(s) négatif(s) :
Conclusion :

3.0.17 Adaptative Physics Based Tetrahedral Mesh Generation Using Level Sets

Auteur(s) : Robert Bridson, Joseph Teran, Neil Molino, Ronald Fedkiw.
Publication : Engineering with Computers (2005)
Sujet(s) abordé(s) :
Présentation d'un algorithme de génération de Mesh Tetrahédral, dont l'entrée est une fonction distance, une grille cartésienne ou un octree. **Principe** :
Point(s) positif(s) :
Point(s) négatif(s) :
Conclusion :

4 Mise en place du modèle global

5 Modèle de flamme et de fumée

Comme décris dans l'aperçu, nous nous sommes focalisé sur un modèle "réaliste" qui peut être exploité en temps réel. Pour cela nous avons choisit un modèle de fluide pour représenter les flammes ainsi que pour la fumée.

Le principe de base est le suivant : lorsqu'un objet brûle il libère un fluide inflammable, si la température est suffisante ce gas s'enflamme, sinon il devient en partie de la fumée. Quand le combustible enflammé s'éloigne de la source de chaleur, il refroidi et lorsqu'il ateint une température suffisament faible il se transforme en un autre gas(invisible) et en fumée.

Pour faire cela nous avons besoin de quatre informations en permanance : le champ de vitesse, de température, de combustible et de fumée. En appliquant ensuite le modèle développé par **Jos Stam** dans **Stable Fluid** nous pouvons faire évoluer le système.

5.1 Plus de détails :

Le modéle de fluide qui semble être bien adapté à la représentation du feu est basé sur le principe des équations de Navier-Stokes pour la dynamique des fluides indéformables :

$$\frac{\partial x}{\partial t} = -(\vec{u}.\nabla)x + \nu\nabla^2 x + \vec{f} \tag{1}$$

Ou u représente le champ de vitesse du milieu et x la valeur transporté par ce champ (densité par exemple)

Cette équation est constitué de 3 termes principaux (4 en théorie mais on a pas introduit l'effet de la pression ambiante) :

- Un terme d'advection : $-(\vec{u}.\nabla)x$
  Le champ de vitesse va servir à transporter la quantité de fluide dans

l'espace. L'auto-advection du champ de vitesse ($-(\vec{u}.\nabla)\vec{u}$) est ce qui va permettre le mouvement des particules de gas dans le milieu.

- Un terme de Diffusion : $\nu\nabla^2 x$
  Le terme ν représente la viscosité du fluide et le terme de diffusion de l'équation de Navier-Stokes sert en fait à modéliser la capacité d'un fluide à se mouvoir dans l'espace qui l'entoure. Plus ν est élevé et moins le fluide pourra se mouvoir dans le milieu.

- Un terme représentant les forces extérieures : $\vec{f}$
  Ce terme sert a modifier le comportement de la valeur x dans le champ de vitesse lorsqu'une force extérieure rentre en jeu (du vent par exemple).

Pour construire notre modele de feu, nous allons considerer 2 variables : $\vec{u}$ le champ de vitesse et ρ la densité du fluide. Notre feu sera alors un champ de densité en mouvement dans un champ de vitesse.

La résolution se fera par étapes : l'ajout des forces externes, la diffusion du fluide et l'impacte de l'advection.

1. Les forces :
   L'ajout des forces externes dans le cas de la densité consiste en fait a ajouter une souce de densité la ou on le souhaite.

2. La diffusion :
   Cela consiste à échanger la densité d'une case de notre grille de l'espace avec ses 6 voisins (haut, bas, gauche, droite, devant, derriere). Ce qu'il se passe en fait c'est que la case va perdre de la densité en la cédant à ses voisins mais va aussi en gagner car ses voisins aussi sont soumis a cela et lui transfére aussi du fluide.

   Résolution :
   $$\frac{\partial \vec{u}}{\partial t} = \nu\nabla^2\vec{u} \tag{2}$$
   $$\vec{u}(\vec{x}, t+\delta t) = \vec{u}(\vec{x}, t) + \nu\delta t\nabla^2\vec{u}(\vec{x}, t) \tag{3}$$
   D'où :
   $$(I - \nu\delta t\nabla^2)\vec{u}(\vec{x}, t+\delta t) = \vec{u}(\vec{x}, t) \tag{4}$$

   Calcul du laplacien :
   $$\nabla^2 p = \frac{p_{i+1,j,k} - 2p_{i,j,k} + p_{i-1,j,k}}{(\delta x)^2} + \frac{p_{i,j+1,k} - 2p_{i,j,k} + p_{i,j+1,k}}{(\delta y)^2} + \frac{p_{i,j,k+1} - 2p_{i,j,k} + p_{i,j,k-1}}{(\delta z)^2} \tag{5}$$
   Qui dans le cas où $\delta x = \delta y = \delta z$ nous donne :
   $$\nabla^2 p = \frac{p_{i+1,j,k} + p_{i-1,j,k} + p_{i,j+1,k} + p_{i,j-1,k} + p_{i,j,k+1} + p_{i,j,k-1} - 6p_{i,j,k}}{(\delta x)^2} \tag{6}$$

Grace a cela on trouve facilement la formule pour la résolution :

$$u_{i,j,k}^{l+1} = \frac{u_{i+1,j,k}^{l} + u_{i-1,j,k}^{l} + u_{i,j+1,k}^{l} + u_{i,j-1,k}^{l} + u_{i,j,k+1}^{l} + u_{i,j,k-1}^{l} + \frac{(\delta x)^2}{\nu \delta t} u_{i,j,k}^{l}}{6 + \frac{(\delta x)^2}{\nu \delta t}} \tag{7}$$

La résolution linéaire de cette équation combinée à une relaxation de Gauss-Seidel nous permet de considerer une méthode stable, même pour des taux de diffusion ($\frac{\nu \delta t}{(\delta x)^2}$) grand.

3. L'advection :
   L'advection est le procédé par lequel le champ de vitesse du fluide transporte la matière d'un endroit a un autre. Le plus simple est de voir la densité de matiere comme une quantité de particules et qu'il faut voir a chaque pas de temps. Et ce que l'on fait c'est que l'on considére les centres de nos cellules comme étant des particules, et plutot que de regarder ou vont se rendre ces particules, on regarde d'ou elle proviendrait si on faisait un pas de temps en arriere. On interpole ainsi les densités depuis leur position d'avant avec celles de leurs plus proches voisins.

5.2 Nos améliorations :

Nous avons rajouté des éléments en plus dans notre code nous permettant de gérer plusieurs phénoménes pour notre feu :

5.2.1 La Température :

La température est un facteur important pour le feu car la température va générer une force qui va modifier le champ de vitesse (l'air chaud qui monte) et donc cela va avoir un impact sur le mouvement de notre flamme.
Cette force a pour forme :

$$\vec{f} = \sigma T \vec{k} \tag{8}$$

Nous avons donc un paramétre de température en plus de la densité qui est géré par les mêmes équations, mais a cela nous avons rajouté un coefficient de refroidissement qui fait baisser la température lorsque l'on s'éloigne de la source.

5.2.2 Vorticity confinement :

Le fait d'utiliser une méthode de relaxation pour rendre le système stable a pour conséquence de provoquer une dissipation d'énergie dans notre calcul et donc une perte de donnée.

Le moyen de palier à ce problème est d'utiliser la technique de Fedkiw : "vorticity confinement" qui redonne au fluid l'energie perdue par le calcul du laplacien.

5.2.3 Génération de fumée :

Le feu dans notre modèle produit de la fumée, nous avons donc géré un deuxieme fluide évoluant dans le même champ de vitesse avec des paramétres qui lui sont propres.

Nous avons donc introduit un paramétre de **consommation de la matiere** qui a chaque pas de temps va diminuer la densité de fluide "feu". En paralléle nous produisons de la densité "fumée" la où on consume le feu , modulé avec un paramétre de convertion (ex : 10 particules de feu forment 3 particules de fumée).

5.3 Méthode de rendu

5.3.1 Affichage volumique

Dans notre cas nous avons à faire à des textures volumiques donc nous nous sommes confrontés très tôt au probléme de l'affichage. Nous avons donc opté pour la solution suivante :

Afficher N plans de coupe de notre texture. Chaque plan étant orthogonal a la direction de la caméra. Pour cela nous avons du implementer toutes les actions qui nous permettaient de nous déplacer dans l'espace autour d'un point et donc de faire pivoter nos plans de coupe en meme temps que la caméra. Ceci nous permet alors de visionner notre texture volumique depuis n'importe quel poit de l'espace.

5.3.2 Rendu de la flamme

Une flamme ne saurait être jolie si l'on ne prend pas en compte la variation lumineuse en fonction de la température. Pour cela nous avons considéré notre flamme corps un modéle de corps noir, c'est à dire une entité qui rayonne à des longueurs d'ondes différentes en fonction de la température, cela en utilisant la formule de Planck pour le corps noir :

$$L_\lambda = \frac{2hc^2\lambda^{-5}}{exp(\frac{hc}{k\lambda T}) - 1} \tag{9}$$

Où h est la constante de planck, c la vitesse de la lumiere, et k la constante de Boltzmann.

On obtient donc une relation entre la température de notre flamme et sa luminance, ce qui nous permet d'avoir un rendu fonction de la température. De plus nous avons ajusté la transparence de la flamme en réglant l'α en tenant compte de la densité de gas qui brule et de la température de la flamme (les flammes bleus sont plus transparentes que les rouges et jaunes).

5.3.3 Rendu de la fumée

Pour le rendu de la fummée, il s'agit simplement d'un niveau de gris avec une transparence α fonction de la densité de fumée en un point.

5.3.4 Multitexturing

Le second probléme d'affichage qui se posa à nous fut le moment ou nous devions afficher la flamme ET la fumée. Le fait d'afficher sous forme de plan posa des probleme au niveau de la précision du Z-Buffer d'OpenGL qui ne savait plus quelle texture était devant l'autre et cela provoquait un effet de clignotement indésirable.

La solution que nous avons retenue pour palier a ce problème était de réaliser un fragment shader au niveau du GPU pour faire un affichage Multitexture prenant en compte l'alpha des 2 texture.

Le principe en lui même est assez simple :
Pour chaque pixel de notre texture
si $\alpha_{texture1} > \alpha_{texture2}$ alors
affichage texture1 ;
sinon
affichage texture2 ;
fin

5.3.5 amélioration de rendu : Bruit de Perlin 4D

Dans la réalité il y a toujours un "vent ambiant" autour d'une flamme, ce qui à pour conséquence de multiples effets de mouvements au niveau de sa surface. Pour simuler cet effet nous ajoutons **lors du rendu par le fragment shader** un bruit de perlin 4D (dont une dimension temporelle pour conserver la cohérence du mouvement).
Le fait d'ajouter ce bruit lors du rendu dans le shader ne modifie pas la position spatiale des densités, des températures et de la fumée et ne perturbe donc pas les calculs de l'équation de dynamique des fluides.

Les rendus obtenus étant plutôt convaincant :

Sans bruit de Perlin :

Avec Bruit de Perlin :

L'expression mathématique du bruit de Perlin est la suivante :
$P_n(x) = \sum_{i=0}^{n-1} p^i bruitPerlin(2^i.x)$.
C'est une série (absolument) convergente, majorée par la série géométrique :
$\sum_{i=0}^{n-1} p^i = \frac{1-p^n}{1-p}$.

5.4 Résultats

Taille de la grille : 20 x 20
FPS : 25

Taille de la grille : 30 x 30
FPS : 8

Taille de la grille : 30 x 30
FPS : 4

6 Modèle de flamme et de fumée GPU

Nous utilisons le langage **GLSL** (Open**GL** **S**hading **L**anguage). Ce langage de shader est utilisé en général pour le rendu de scène 3D. Le shader utilisé est un **fragment shader**. Il est (très grossièrement) destiné à changer la couleur d'affichage d'un objet.
D'autres outils de programmation existent (comme OpenCL) mais nous n'étions pas tous compatibles avec cette technologie. Nous avons donc choisi le GLSL qui est suporté par un bien plus grand nombre de carte graphiques.

Nous avons donc du mettre en place des outils permettant de résoudre notre système de fluide avec des outils qui sont à priori pas destinés à ce genre de problèmes.

6.1 Transformation du problème pour une résolution GPU

6.1.1 Matrice 3 dimensions == Texture 3D

Notre matrice 3 dimensions dans lequel nous traitons le modèle de fluide peut se rapprocher d'une **Texture 3D** d'OpenGL.

6.1.2 Operation sur le point (i,j,k) de la texture == FragmentShader

Un shader c'est un programme. Un fragment shader est un programme qui s'applique sur la couleur d'un pixel d'un objet.

Le principe de fonctionnement d'un shader est simple. On active le shader, on dessine, on désactive le shader. Tout ce qui est dessiné pendant qu'il est actif subit toutes les modifications effectuées par le shader.

Ainsi lorsqu'on fait une triple boucle pour parcourir l'ensemble des elements du tableau, il suffit d'appliquer un shader sur notre texture3D et toutes les couleurs de celle-ci seront modifiées.

La première difficulté est de pouvoir traiter l'ensemble de la Texture 3D avec le shader. On peut le faire en dessinant plusieurs fois un plan et en changeant ses coordonnées de texture.

6.1.3 Iteration == Framebuffer (render to texture 3D)

Un **shader** ne s'applique qu'au moment du dessin. C'est à dire qu'on active le shader, on dessine, on désactive le shader. Tout ce qui est dessiné lorsqu'il est actif subit ses modifications. Mais du coup, la texture 3D initiale n'est jamais modifiée. Or lorsqu'on fait une itération dans le calcul il est essentiel de conserver le résultat dans une texture 3D.

Avec une méthode de **render to texture** il est possible d'enregistrer ce qui s'affiche à l'écran dans une texture.

Principe du render to texture 3D Nous avons un **FrameBuffer**. Lorsqu'il est actif, le dessin ne s'affiche pas sur l'écran, mais est conservé en mémoire. Nous pouvons rediriger la sortie du framebuffer vers une texture.

Il nous faut donc deux textures : la texture courante, et la texture dans lequel on enregistrera les résultat du traitement par le shader.

Si notre texture 3D fait 128 x 128 x 128, nous allons dessiner dans une fenetre de 128x128 chaque tranche de la texture.
Lors du traitement de la tranche i, nous :

- activons le shader (qui applique le calcul de résolution) (en fait on l'active une seule fois au début)
- redirigeons la sortie du framebuffer vers la tranche i de la texture de sortie,
- dessinons un plan (dont les coordonnées de texture correspondent à la tranche i) et dont la texture est la texture d'entree

Ainsi en sortie de l'itération nous avons dans la seconde texture le résultat du shader appliqué à la premiere texture.

Remarque importante : Sur le papier cette astuce est très compréhensible. La mettre en place n'est pas si facile puisqu' un simple paramètre mal configuré entraine un résultat faux, et trouver la cause n'est pas évident...

6.2 Amélioration

Un framebuffer est capable de dessiner dans plusieurs tranches de la texture en même temps (en général 8). Par défaut il dessinera la même chose dans les 8 tranches. Nous avons donc créé un shader qui dessine dans les 8 tranches de sortie les 8 tranches de l'entrée correspondantes. Ainsi nous évitons les allez et retour entre le CPU et le GPU ce qui permet un gain en performances.

6.3 Résultats

7 Propagation sur l'objet

8 Destruction de l'objet

9 Propagation dans l'environnement

10 Références