Travaux dirigés de Simulation n°1

Cours de simulation et animation

—IMAC troisième année—

Introduction à la simulation de fluide et advection

Objectifs

- Présentation de la simulation de fluide et motivations,
- Début de la simulation.

Programme

- Introduction du projet,
- Découverte de l'outil fourni et du modèle de grille,
- Fonctions d'intégration et la déplacement des particules,
- Advection de la couleur et de la vélocité.

▸ Généralités et consignes.

Enseignants

TDs
Nadine Dommanget, **dommange@univ-mlv.fr**
igm.univ-mlv.fr/ dommange/enseignement/opengl.html
Bureau **4301** à l'ESIEE (4ème épi, 3ème étage, 1ème porte)

CM
Eric Incerti, **eric.incerti@univ-mlv.fr**

Organisation

Les TDs sont répartis en quatre séances de deux heures.
Ils sont à réaliser en binômes/trinômes sur les ordinateur de la salle **1B077** ou sur un ordinateur personnel. Le projet sera tenu de fonctionner sur les ordinateurs de l'Université.

L'objectif est de disposer à leur issue d'une simulation de fluides 2D de type Eulerienne (basée sur une grille).
De manière à faciliter et à focaliser le travail sur la simulation, une base de code est fournie (permettant notamment le stockage des données, les outils d'interpolation et la visualisation). Il n'est pas obligatoire de l'utiliser, tant que l'esprit des TDs et du projet est respecté.

A l'issue des séances, le code réalisé sert de base au projet. Les parties du projet à caractère artistiques ou suffisament independantes de la simulation peuvent être débutées dès à présent.

Le projet consiste à faire apparaître une forme dans un fluide.
Les modalité d'apparition et d'animation sont libres, de même que le style du rendu, l'utilisation de la 2D ou 3D, la recherche thématique/scénaristique...

L'objectif de ces libertés est de vous permettre de vous spécialiser sur un ou plusieurs des aspects qui vous sont les plus chèrs (dessin, scénario, rendu, 3D, interactivité, richesse des effets d'animation...). L'absence de solution précise au problème posé (de l'apparition de la forme) vise à provoquer votre créativité sur l'infinié d'effets visuels permis par

la simulation physique. Le réalisme de la physique n'est pas une contrainte, tant que le résultat est contôllé par une intention et évocateur d'un fluide.

Notation

La notation est exclusivement basée sur le projet. Elle s'effectuera sur la base du code final, d'un mini-rapport (quelques pages) et surtout d'une soutenance/demo.

Programmation

Le template est basé sur celui sur lequel vous avez travaillé en OpenGL en IMAC2. Il combine SDL et OpenGL3 dans une architecture basique orientée objet en C++.
L'archive (`TDsSimu.tar.gz`) est téléchargeable sur mon site (adresse en haut), en compagnie les slides et de ce sujet.

Le template n'étant pas complet par rapport au temps imparti au TPs (et à la quantité de code à écrire...), des éléments à ajouter au template seront proposés en plus au cours des séances. Notamment, il conviendra en temps utile d'ajouter une bibliothèque d'algèbre linéaire notamment pour les opérations vectorielles et matricielles de grande ampleur et pour la résolution d'un système.

▸ **Références.**

Elles sont extrêmement nombreuses (quoique généralement difficiles).
Les méthodes vues en TPs sont toutes extraites du très bon livre :
"**Fluid Simulation for Computer Graphics**", *Robert Bridson* : http://www.cs.ubc.ca/ rbridson/fluidbook/

▸ **Exercice 1. Découverte**

A - Partie importante du code

L'essentiel de ce qui est relatif à la simulation a lieu dans la classe `Simulation` dont une instance est stockée par `application`.

B - Objectif de la simulation de fluides

On utilise les simulations de fluides pour réaliser l'animation de liquides ou de gaz, en physique comme en synthèse d'image. En physique, l'exemple le plus courant est la prévision météorologique. En synthèse d'image, les exemples les plus courants sont l'animation de l'eau et la fumée, notamment pour le cinéma.

En pratique, la simulation de fluides ne consiste en rien de plus que l'évaluation de l'évolution de la vitesse d'un matériau fluide sur une zone de l'espace et dans le temps.

C - Simulation Eulerienne

Dans le cadre d'une simulation **Eulerienne**, on évalue en fait les valeurs de la vitesse en des positions fixes dans l'espace. De même, on peut évaluer en ces points fixes d'autres grandeurs physiques comme la couleur, température, pression...

Les points fixes sont, pour plusieurs raisons (et notamment de calcul différentiel), traditionnellement ceux d'une grille régulière de cellules carrés (en 2D) ou cubiques (en 3D).

Pour visualiser le résultat de la simulation, on peut afficher les vecteurs vitesse et autres grandeurs sur les points de la grille, mais il est généralement plus compréhensible, esthétique et efficace de disposer des particules sur l'espace de la grille et de les faire se dépacer et changer en fonctions du champ de vélocité, de couleurs... Ainsi on voit effectivement le fluide circuler. Toutefois, il ne faut pas confondre une simulation **Eulerienne** rendue par des particules, avec une simulation de type **Lagrangienne**, qui elle utilise réellement des particules qui interagisse entre elles pour la partie simulation.

La méthode **Eulerienne** est généralement plus difficile mathématiquement, moins intuitive et peu efficace.

Pourquoi donc l'avoir choisie par rapport à une méthode "tout particules" ?

Elle est plus fiable, donne les résultats les plus justes et les plus esthétiques. Elle est très largement la plus utilisée pour les fluides dans le monde de la physique comme en synthèse d'images.

D - Méthode non temps-réel

Il faut toutefois bien prendre conscience que le temps réel est généralement inaccessible par cette méthode (notamment en 3D).
Enregistrer les résultats pour chaque frame est donc une bonne approche.

On peut par exemple évaluer l'image finale de chaque pas de temps puis rejouer la vidéo des images pour voir l'animation à la bonne fréquence. Cette fonctionnalité est prévue dans le template. Pour générer une image à chaque frame : décommenter `this->storeFrame();` ligne 555 de Application.cpp. Pour créer la vidéo à partir des images enregistrées puis les supprimer, lancer le script shell video : `sh video nomDeLaVideo`. nomDeLaVideo.mp4 est à présent disponible dans le dossier `../videos/`. Cette fonctionalité est à utiliser précautioneusement pour ne pas remplir votre disque d'images (si vous laissez tourner l'appli longtemps ou ne supprimez jamais les images), ou de vidéos.

Sinon, on peut aussi stocker les différents champs de données à chaque pas de temps, puis faire leur lecture dans une application interactive, avec la visualisation par particules. Decoupler la simulation de la visualisation peut notamment permettre d'accélérer la simulation en parallélisant sur plusieurs machines ou sur la carte graphique (non occuppée par le dessin à ce moment là). Avoir enregistré la simulation sur le disque (champs de données ou particules de visualisation) peut également permettre d'utiliser un moteur de rendu offline pour embellir le fluide final. Cette fonctionalité n'est pour l'instant pas incluse au template, mais libre à vous de la réaliser si le besoin se présente.

E - Visite et expériementation

Depuis le main, vous pouvez changer les principaux paramètres de la simulation en devenir. L'implématation 3D n'est pas disponible, ne changez donc pas "surface".

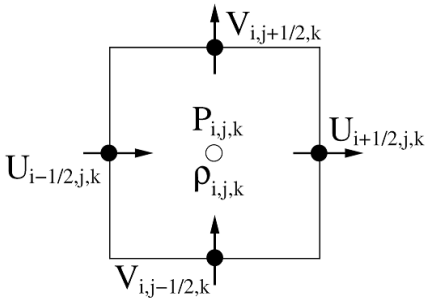
a) Faites varier les paramètres de taille et de discrétisation de la grille.

b) Faites varier la densité des particules. Comment sont-elles réparties sur la grille ?

c) Commentez les lignes 217 à 222 puis réaffichez les différents éléments un par un pour bien les identifier. Mettez-les en correspondances avec les différents tableaux stockés dans `Simulation`.

d) Où sur la grille sont evaluées la couleur et la vélocité ?

Les données sont réparties selon une **MAC-grid**, illustrable ainsi, avec U et V respectivement les composantes horizontales et verticales de la vitesse :



e) Attachez-vous à comprendre les fonctions d'interpolation bilinéaire dans le cas de données stockées aux centres des cellules, ou bien décomposées sur les bords.

f) Visitez le reste du code pour comprendre au maximum les étapes principales de tout ce qui s'y passe à chaque frame.

▸ Exercice 2. Schémas d'intégration

A - Forward Euler

a) A partir d'une position courante, d'un vecteur vitesse en cette position et de la durée d'un pas de temps dt, comment calculeriez-vous la position à $t + dt$?

b) Mettez votre solution dans la fonction `forwardEuler1stOdrTimeIntegration(...)`.

c) Appellez cette fonction dans `integrate(...)`. Faites tout le cheminement de code pour comprendre l'animation des particules.

B - Runge-Kutta ordre 2

La solution précédente n'est exacte que dans le cas où la vitesse sur le chemin entre le point de départ et d'arrivée est constante et égale à celle de départ.

a) Faites mieux en interpolant une nouvelle vitesse au point atteint après dt, puis en utilisant plutôt cette vitesse intermédiaire dans un schéma forward Euler.

b) Mettez votre solution dans la fonction `rungeKutta2ndOdrTimeIntegration(...)`.

c) Appellez plutôt cette fonction dans `integrate(...)`. La différence n'est pas visible dans ce cas mais la simulation est plus exacte avec ce deuxième schéma.

C - Runge-Kutta ordre 3

a) Le schéma Rung-Kutta ordre 3 est encore meilleur. Lisez la fonction pour en comprendre le principe global.

b) Appellez plutôt cette fonction dans `integrate(...)`. Encore une fois les bons résultats de la méthode ne sont pas forcément mis en valeur par les paramètres de la simulation que vous utilisez (pas de temps, vitesse, variation spatiale de la vitesse...).

▸ Exercice 3. Comportement aux limites

A - Solid walls

Le long d'un solide, la composante orthogonale de la vitesse d'un fluide est nulle (le fluide ne peut pas rentrer dans le solide).

a) Modifier l'initialisation des composantes des vitesses pour satisfaire ce principe lorsque le booléen solidWalls est activé.

b) Testez la différence.

c) Testez aussi sur l'autre initialisation proposée pour les vitesses (deux lignes à decommenter dans la même fonction).

d) Désactivez le booléen solidWalls pour la suite.

▸ Exercice 4. Advection

Dans une simulation **Eulerienne**, l'un des principaux challenge est l'advection, c'est à dire la conservation des propriétés du fluide lors de son déplacement sur la grille (exemple : couleur, température et même vitesse). Il s'agit pourtant d'un problème trivial en résolution **Lagrangienne**, comme chaque particule stocke de manière fixe ses propres informations.

Nous allons nous attacher à réaliser l'advection de la couleur, puis de la vélocité en utilisant une méthode très intuitive et stable appellée "**semi-Lagrangienne**".

Le principe est le suivant.

- On trouve la vélocité au centre de la cellule (là où la couleur doit être advectée à chaque pas de temps).
- On imagine ensuite une particule virtuelle arrivant en ce point à t.
- On intègre la position à $t - dt$ de la particule en utilisant la vélocité. La position obtenue correspond à l'endroit où devait se trouver la particule virtuelle à $t - dt$.
- On interpole la couleur en cette position.
- On affecte cette couleur au centre de la cellule.

A - Advection de la couleur

a) Réalisez l'advection de la couleur dans la fonction `advectColors(GLfloat dt)`.

b) Verifiez que la fonction est bien appellée.

c) Pour voir la différence, l'animation des particules doit avoir lieu. On la percoit mieux sur le deuxième modèle de vitesses initiales (mode diagonal et non circulaire). Que constatez-vous ?

d) Faites également une mise à jour des valeurs de couleur des samples sur le GPU à la fin de la fonction (inspirez-vous de la fin de la fonction `advectParticles(GLfloat dt)`). Désactivez l'affichage des particules et vérifiez que l'advection sur les couleurs des particules est bien visible.

e) Observez en mode circulaire que les couleurs finissent par se mélanger. Il s'agit d'un phénomène de diffusion qui est l'un des problème de la simulation `Eulerienne`. Il est du aux interpolations bilinéaires successives qui font perdre de la précision et créée à la longue une erreur se traduisant comme un flou.

f) Quelle est selon vous la solution naturelle pour résoudre ce problème ? Expérimentez-là. Les résultats devraient vous paraître satisfaisant pour notre exigence en précision (assez basse...).

B - Advection de la vélocité

Pour résoudre les équations de dynamique des fluides de Navier-Stokes ou Euler (les mêmes mais sans la viscosité), il est nécessaire de réaliser une advection sur les vitesses également. Nous étudierons les formules la prochaine fois, mais en attendant, vous pouvez déjà réaliser votre advection sur les velocités, en copiant sur le modèle de celle des couleurs.

A - Réalisez l'advection de la vitesse dans la fonction `advectVelocities(GLfloat dt)`. Il faut bien mettre à jour les composantes directionnelles de la vitesse `velocitiesX` et `velocitiesY`.

B - Verifiez que la fonction est bien appellée.

C - Pour voir la différence, l'animation des particules doit avoir lieu. On la percoit mieux sur le deuxième modèle de vitesses initiales (mode diagonal et non circulaire). Que constatez-vous ?

D - Faites également la mise à jour nécesaires des valeurs de velocité des bordures de cellules sur le GPU. Désactivez l'affichage des particules, puis les velocités aux centres des cellules et vérifiez que l'advection sur les velocités des particules est bien visible (aux bords).

E - Observez en mode circulaire le comportement lorque vous réactivez les solidWalls. Expliquez le phénomène. Il devrait normalement se résoudre avec la suite des étapes de la simulation que nous verrons aux prochaines séances.

▸ **Exercice 5. En préparation du projet**

Cet exercice n'est pas à réaliser pendant les séances. Voyez-le comme un moyen de vous lancer au plus vite sur la direction artistique et les outils externes de votre projet.

A - Des idées pour la forme ?

On a vu que les vitesses initiales permettent de construire des dynamiques de déplacement des particules, et donc potentiellement des formes. De même, en changeant les conditions aux bords, on a conditionné le sens et les limites du déplacement.

a) Réfléchir à des pistes utilisant ces résultats pour afficher une forme.

b) Les essayer en initialisant/modifiant les vélocités dans cette idée.

B - Charger une forme/animation.

a) Trouvez un format pour décrire la forme que vous voulez créer (images PPM par exemple).

b) Essayez de charger un champ de velocity en vous basant sur votre fornat.

C - Dessinez une forme/animation.

Commencez à réflechir à la forme/animation que vous prévoyez de faire pour votre projet. Si votre intérêt est plus artistique que purement technique, il faudra soigner cette partie et penser à un design recherché et à une scénarisation.