TP1 – Simulation de Fluide

Programmation Avancée de Jeux Vidéos – Nicolas Hurtubise

Contexte

Le programme fourni est une simulation de particules avec la méthode SPH (*Smoothed-particles hydrodynamics*).

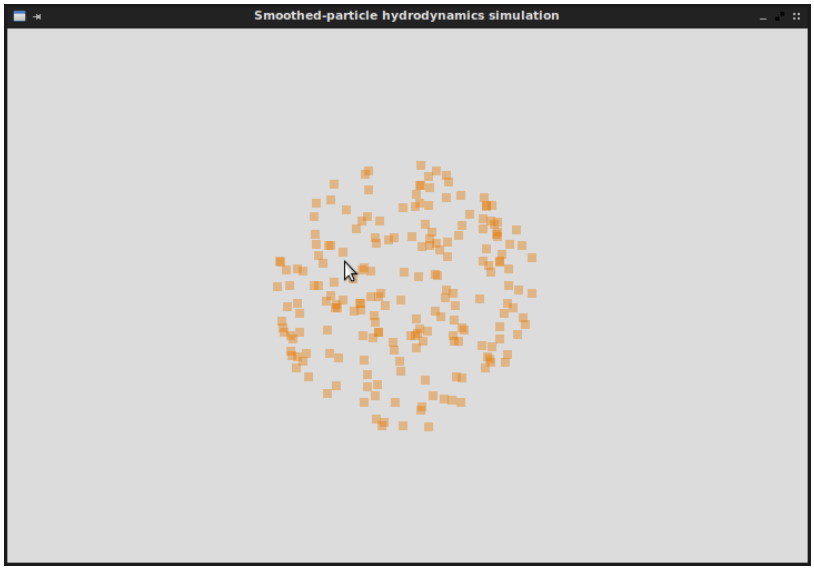
Le code est dérivé de l'implantation sous licence MIT par Lucas V. Schuermann, qui explique en détails l'idée derrière une simulation de fluides SPH sur son blog. La simulation présentée sur ce blog est gardée simple pour des fins pédagogiques... Ce qui limite beaucoup le nombre de particules qu'on peut y ajouter!

Votre tâche sera :

1. D'optimiser ce code pour pouvoir le faire rouler avec un nombre aussi grand que possible de particules
2. D'ajouter une petite fonctionnalité pour mettre en pratique le design pattern *Command*

Interface graphique

La simulation est affichée dans une fenêtre SDL :



Les contrôles sont les suivants :

- *Flèches (haut/bas/gauche/droite)* : change la gravité de direction
- *Numéros 1 à 9* : relance la simulation avec 1 à 5000 particules
- *Barre espace* : fait “exploser” le fluide en donnant une vitesse aléatoire aux particules
- *Touches A, S, D* : change le mode d’affichage de la simulation
- *Clic droit* : ajoute un bloc de particules
- *Clic gauche* : ajoute une seule particule

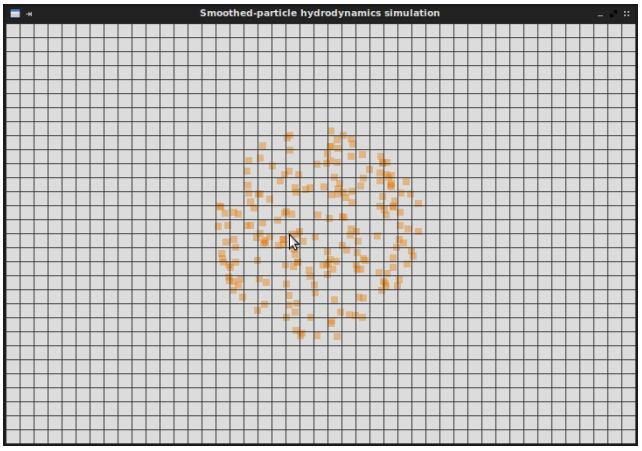
1 - Optimisations

Optimisation principale

Le plus gros problème de ce code est clair : calculer la densité, la pression et les forces totales exercées en chaque point n'est pas fait efficacement.

Pour chaque particule, le code considère toutes les autres pour calculer l'impact de chacune, même si les particules trop éloignées n'interagissent pas. On peut accélérer de beaucoup la simulation en utilisant une **grille spatiale** qui contient une liste de pointeurs vers des particules dans chaque région.

Si on considère l'espace 2D de la simulation, on peut le discrétiser en cases de taille $Nb_{pixels} \times Nb_{pixels}$:



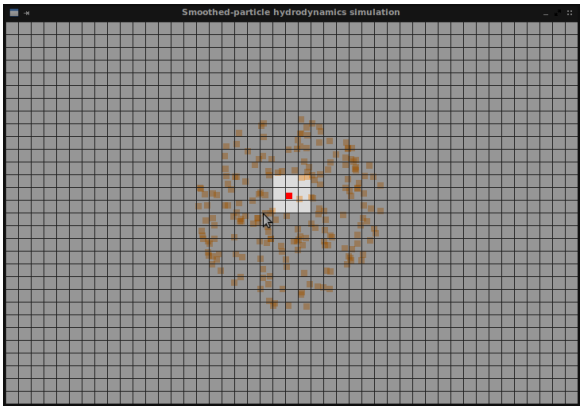
La distance maximale à laquelle deux particules peuvent s'entre-influencer est définie par taille du kernel H :

```
const double H = 16.f; // kernel radius
```

Si on choisit une taille pour les cellules $Nb_{pixels} = H$, on peut alors se limiter à appliquer le calcul des densités, pressions, etc. en considérant seulement les particules qui se trouvent :

- Dans la **même cellule** de la grille
- Dans une **cellule voisine** de la grille

Pour la particule affichée en rouge ici, on peut se limiter à vérifier les interactions dans les 9 cellules mises en évidence ici :



Implantation

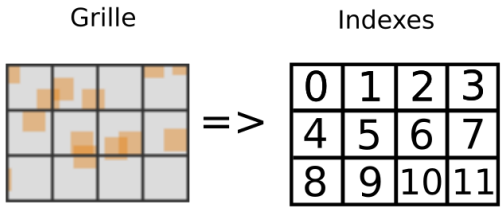
Vous **devez** utiliser la structure de données demandée :

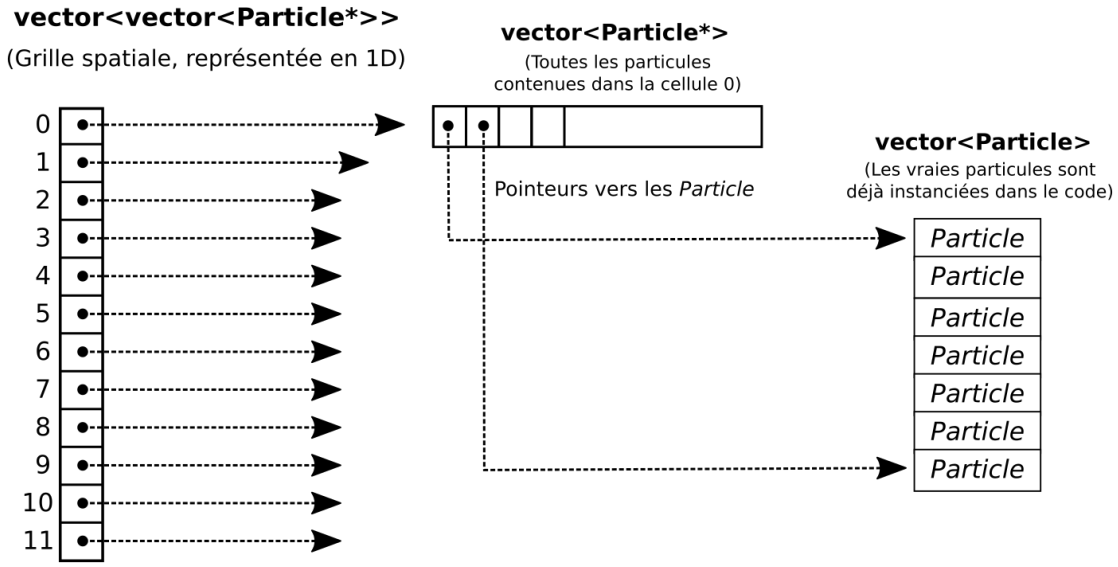
> Un `vector` (à une seule dimension) qui contient dans chacune de ses cases un `vector` de pointeurs vers des particules.

En C++, ça donne :

```
std::vector<std::vector<Particle*>> grid;
```

Si on visualisait cette structure de données sur une plus petite grille, on aurait quelque chose comme ça en mémoire :





Prenez la méthode :

```
ParticleManager::update(unsigned long dt)
```

et ajoutez-y au tout début une première étape où vous construisez la grille.

Vous pourrez ensuite utiliser cette structure de données dans les fonctions appropriées :

```
ParticleManager::computeDensityPressure()
ParticleManager::computeForces()
```

Rendu de la grille à l'écran

Pendant que vous développez votre grille, assurez-vous qu'elle fonctionne correctement en l'affichant à l'écran.

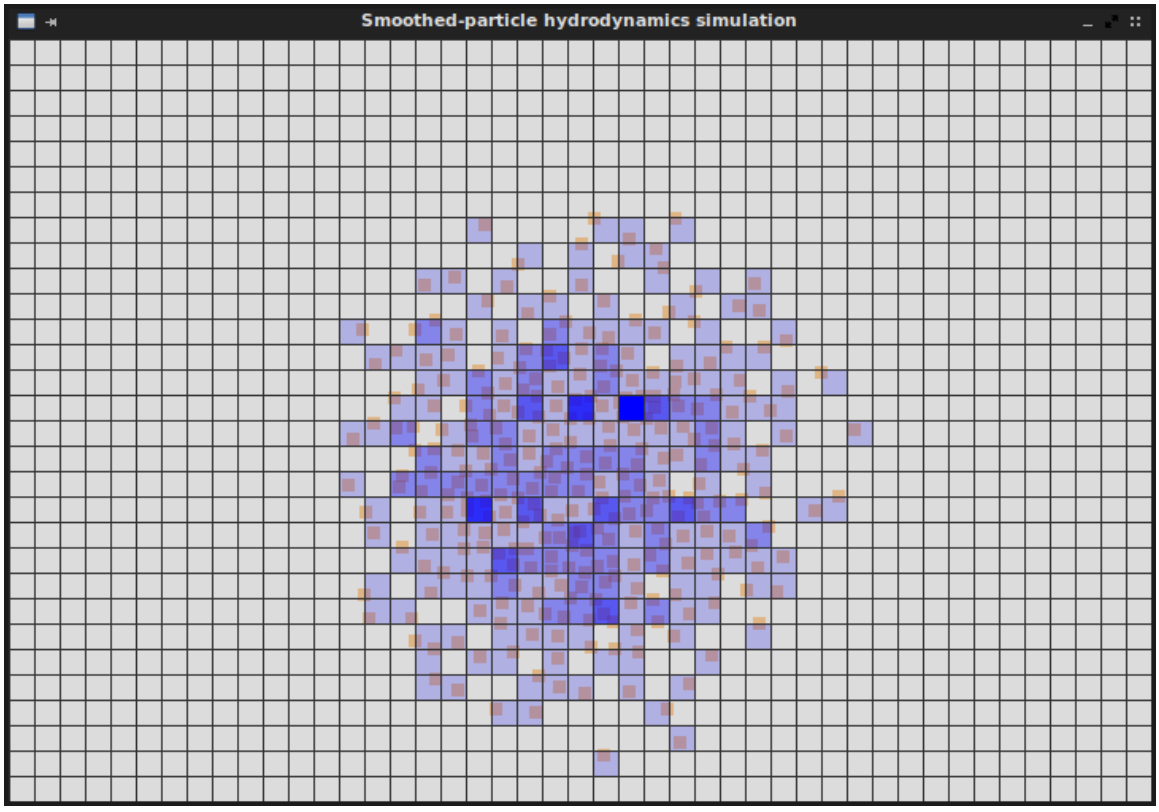
Complétez les deux fonctions :

```
1. void ParticleManager::renderGrid(SDL_Renderer* renderer)
2. void ParticleManager::renderCells(SDL_Renderer* renderer)
```

qui doivent respectivement

1. Afficher les lignes de la grille
2. Visualiser le nombre de particules dans chaque cellule en dessinant la cellule avec un bleu plus ou moins opaque

Le résultat devrait avoir l'air de :



Vous pouvez changer le mode de rendu appelé en utilisant les touches `A`, `S` et `D`.

Optimisations secondaires

Un coup parti, on peut faire d'autres modifications au code, qui vont avoir plus ou moins d'impact.

Voici quelques possibilités :

1. Remplacer les appels à `pow(...)` dans le fichier `ParticleManager.cpp` par des expansions manuelles, par exemple :

   `pow(x, 3.0)` est équivalent à `x * x * x`

2. Remplacer les appels à `pow(...)` dans le fichier `ParticleManager.h` par des expansions manuelles (même chose que 1, mais dans l'autre fichier)

3. Remplacer les calculs des constantes utilisées dans le code par un précalcul (ajouter des constantes supplémentaires au lieu de faire la multiplication dans le code directement) :

   - `MASS * POLY6` dans `computeDensityPressure()`
   - `MASS * SPIKY_GRAD` dans `computeForces()`
   - `VISC * MASS * VISC_LAP` dans `computeForces()`

4. Remplacer la comparaison du `renderMode` en `string` par une comparaison d'`enum class`

5. La constante `HSQ` a été commentée dans `ParticleManager`, mais on pourrait s'en servir pour éviter de calculer une racine carrée à certains endroits du code :

   `if(sqrt(x) < H) {...}` est équivalent à `if(x < H*H) {...}`

2 - Fonctionnalité à ajouter

En plus d'optimiser le programme, on vous demande d'ajouter la fonctionnalité **undo/redo** au code.

- `Ctrl-Z` (undo) doit **annuler** la dernière opération faite
- `Ctrl-Shift-Z` (redo) doit ré-appliquer l'opération qui vient d'être annulée
- On doit pouvoir faire plusieurs *Undo* de suite, puis tous les refaire avec des *Redo*
- Si on fait une longue séquence de *Undo* et qu'on fait une *nouvelle opération* par la suite, ce n'est pas possible de faire un *redo*

Les opérations qui peuvent être *undone/redone* sont les suivantes :

- Un clic gauche ajoute un bloc de nouvelles particules à la simulation **(déjà codé)**
- Un clic droit ajoute une seule nouvelle particule à la simulation **(déjà codé)**
- Appuyer sur la touche **C** au clavier doit changer la couleur des particules pour une couleur choisie au hasard **(à ajouter)**

On va discuter en groupe de où et comment ajouter des classes appropriées dans le code.

Vous êtes autorisés à vous inspirer de http://gameprogrammingpatterns.com/command.html au besoin.

Rapport (en PDF seulement)

En plus de votre code optimisé, vous devez remettre un rapport (au format **PDF**, **pas** Word) qui répond aux questions suivantes :

1. **Avant de modifier le programme**, combien de particules arrivez-vous à ajouter avant d'obtenir le premier message de lag dans la console? (Comparez en mode `Debug` **et** en mode `Release`)
2. **Après avoir fait l'optimisation principale seulement (la grille)**, combien de particules arrivez-vous à ajouter avant d'obtenir le premier message de lag dans la console? (`Debug` **et** `Release`)
3. **Après avoir fait toutes les optimisations**, combien de particules arrivez-vous à ajouter avant d'obtenir le premier message de lag dans la console? (`Debug` **et** `Release`)

Pour chaque optimisation (pour la grille et pour chaque optimisation secondaire), donnez une petite analyse de l'optimisation :

- **Avant de faire l'optimisation** : est-ce que cette optimisation vaudrait la peine d'être faite? Justifiez en donnant des captures d'écran du profileur.
- **Une fois l'optimisation faite** : constatez-vous une différence de performance après l'avoir faite? Indiquez **Significative** ou **Non significative** de façon claire
- *Si l'optimisation est significative*, indiquez le nombre de particules qu'elle vous a permis de rajouter à la simulation sans causer de lag

- Quel le coût de cette optimisation en termes de *clarté du code* : à quel point le code est rendu plus difficile à comprendre
- Quel le coût de cette optimisation en termes de *flexibilité du code* : à quel point ça complexifie l'ajout de nouvelles fonctionnalités plus tard

Remise

Vous devez remettre sur Léa :

- Votre code **dans un fichier .zip** (**PAS** au format `.rar`), qui ne contient **pas** les fichiers temporaires de Visual Studio (un dossier nommé `.vs`)
  - Votre fichier `.zip` ne devrait pas prendre plus que ~1Mo, si ça pèse plusieurs dizaines de Mo, il y a un problème
- Votre rapport **au format PDF**

Barème

- 65% : Fonctionnalités demandées implantées correctement
  - (25%) Optimisation principale
  - (15%) Optimisations secondaires
  - (10%) Visualisation de la grille
  - (15%) Undo/Redo des opérations
- 25% : Rapport
  - Réponses aux questions
  - Discussion/analyse des optimisations
- 10% : Qualité du code
  - Code bien commenté, bien découpé en fonctions au besoin, ...

Note sur le plagiat

Le travail est à faire **individuellement**. Ne partagez pas de code, ça constituerait un **plagiat**, et vous auriez *zéro*.

(Sérieusement, la classe est toute petite, je vais m'en rendre compte assez vite si vous échangez du code.)